目　录

上篇　MATLAB 基础篇

下篇　算法程序篇

实 例 目 录

第 2 章　MATLAB 计算基础

第 3 章　MATLAB 程序设计基础

第 4 章 插值

第 5 章 函数逼近

第 6 章 矩阵特征值计算

第 7 章　数值微分

第 8 章　数值积分

第 9 章　方程求根

第 10 章　非线性方程组求解

第 11 章　解线性方程组的直接法

第 12 章 解线性方程组的迭代法

第 13 章 随机数生成

第 15 章 常微分方程的初值问题

第 16 章 偏微分方程的数值解法

第 17 章 数据统计和分析

Part 1

上篇 MATLAB 基础篇

第 1 章　MATLAB 语言概述

经过20余年的补充与完善以及多个版本的升级换代，MATLAB 语言已发展至 R2008A 版本。MATLAB 是一个包含众多科学、工程计算的庞大系统，是目前世界上最流行的计算机软件之一。

1.1 MATLAB 语言的产生与发展

MATLAB 语言的产生是与数学计算紧密联系在一起的。1980 年，美国新墨西哥州大学计算机系主任 Cleve Moler 在给学生讲授线性代数课程时，发现学生在高级语言编程上花费很多时间，于是着手编写供学生使用的 Fortran 子程序库接口程序，他将这个接口程序取名为 MATLAB（即 Matrix Laboratory 的前三个字母的组合，意为“矩阵实验室”）。这个程序获得了很大的成功，受到学生的广泛欢迎。

20 世纪 80 年代初期，Moler 等一批数学家与软件专家组建了 MathWorks 软件开发公司，继续从事 MATLAB 的研究和开发，1984 年推出了第一个 MATLAB 商业版本，其核心是用 C 语言编写的。而后，它又添加了丰富多彩的图形图像处理、多媒体、符号运算以及与其他流行软件的接口功能，使得 MATLAB 的功能越来越强大。

MathWorks 公司正式推出 MATLAB 后，于 1992 年推出了具有划时代意义的 MATLAB 4.0 版本，之后陆续推出了几个改进和提高的版本，2004 年 9 月正式推出 MATLAB Release 14，即 MATLAB 7.0，其功能在原有的基础上又有了进一步的改进，2008 年 3 月推出了 R2008A，它是目前 MATLAB 最新的版本。

MATLAB 经过几十年的研究与不断完善，现已成为国际上最为流行的科学计算与工程计算软件工具之一，现在的 MATLAB 已经不仅仅是一个最初的“矩阵实验室”了，它已发展成为一种具有广泛应用前景、全新的计算机高级编程语言，可以说它是“第四代”计算机语言。

自 20 世纪 90 年代，美国和欧洲的各大学将 MATLAB 正式列入研究生和本科生的教学计划，MATLAB 软件已成为应用代数、自动控制理论、数理统计、数字信号处理、时间序列分析、动态系统仿真等课程的基本教学工具，成为学生必须掌握的基本软件之一。在研究单位和工业界，MATLAB 也成为工程师们必须掌握的一种工具，被认做进行高效研究与开发的首选软件工具。

1.2 MATLAB 的优势与特点

MATLAB 在学术界和工程界广受欢迎，其主要优势和特点有如下几方面。

◆ 友好的工作平台和编程环境

MATLAB 由一系列工具组成，其中许多工具采用的是图形用户界面，包括 MATLAB 桌面和命令窗口、历史命令窗口、编辑器和调试器、路径搜索和用于用户浏览帮助、工作空间、文件的浏览器。这些图形化的工具方便用户使用 MATLAB 的函数和文件。

随着 MATLAB 的商业化以及软件本身的不断升级，MATLAB 的用户界面也越来越精致，更加接近 Windows 的标准界面，人机交互性更强，操作更简单。

同时，MATLAB 提供了完整的联机查询、帮助系统，极大地方便了用户的使用。

MATLAB 简单的编程环境提供了比较完备的调试系统，程序不必经过编译就可以直接运行，而且能够及时地报告出现的错误并进行出错原因分析。

◆ 简单易用的编程语言

MATLAB 语言是一种高级的矩阵语言，它包含控制语句、函数、数据结构、输入和输出和面向对象编程特点。用户可以在命令窗口中将输入语句与执行命令同步，也可以先编写好一个较大的复杂的应用程序（M 文件）后再一起运行。

MATLAB 语言是基于流行的 C++语言基础上的，因此语法特征与 C++语言极为相似，而且更加简单，更加符合科技人员对数学表达式的书写格式。使之更利于非计算机专业的科技人员使用。而且这种语言可移植性好、可拓展性强，这也是 MATLAB 能够深入到科学研究及工程计算各个领域的重要原因。

◆ 强大的科学计算机数据处理能力

MATLAB 是一个包含大量计算算法的集合，其拥有 600 多个工程中要用到的数学运算函数，可以方便地实现用户所需的各种计算功能。

这些函数集包括从最简单最基本的函数到诸如矩阵、特征向量、快速傅立叶变换的复杂函数。

函数所能解决的问题大致包括矩阵运算和线性方程组的求解、微分方程及偏微分方程的组的求解、符号运算、傅立叶变换和数据的统计分析、工程中的优化问题、稀疏矩阵运算、复数的各种运算、三角函数和其他初等数学运算、多维数组操作以及建模动态仿真等。

函数中所使用的算法都是科研和工程计算中的最新研究成果，而前经过了各种优化和容错处理。

在通常情况下，可以用 MATLAB 来代替底层编程语言，如 C 和 C++。在计算要求相同的情况下，使用 MATLAB 的编程工作量会大大减少。

◆ 出色的图形处理功能

MATLAB 自产生之日起就具有方便的数据可视化功能，能够将向量和矩阵用图形的形式表现出来，并且可以对图形进行标注和打印。

高层次的作图包括二维和三维的可视化、图像处理、动画和表达式作图，可用于科学计算和工程绘图。

MATLAB 对整个图形处理功能进行了很大的改进和完善，使它不仅在一般数据可视化软件都具有的功能（例如二维曲线和三维曲面的绘制和处理等）方面更加完善，而且对于

一些其他软件所没有的功能（例如图形的光照处理、色度处理以及四维数据的表现等），MATLAB 同样表现了出色的处理能力。

同时对一些特殊的可视化要求，例如图形对话等，MATLAB 也有相应的功能函数，保证了用户不同层次的要求。MATLAB 还着重在图形用户界面（GUI）的制作上作了很大的改善，对这方面有特殊要求的用户也可以得到满足。

◆ 应用广泛的模块集合工具箱

MATLAB 对许多专门的领域都开发了功能强大的模块集和工具箱。一般来说，它们都是由特定领域的专家开发的，用户可以直接使用工具箱学习、应用和评估不同的方法而不需要自己编写代码。

目前，MATLAB 已经把工具箱延伸到了科学研究和工程应用的诸多领域，如数据采集、数据库接口、概率统计、样条拟合、优化算法、偏微分方程求解、神经网络、小波分析、信号处理、图像处理、系统辨识、控制系统设计、鲁棒控制、模型预测、模糊逻辑、金融分析、地图工具、非线性控制设计、实时快速原型及半实物仿真、嵌入式系统开发、定点仿真、DSP 与通信、电力系统仿真等，都在工具箱（Toolbox）家族中有了自己的一席之地。

◆ 实用的程序接口和发布平台

MATLAB 可以利用 MATLAB 编译器和 C/C++数学库和图形库，将自己的 MATLAB 程序自动转换为独立于 MATLAB 运行的 C 和 C++代码。允许用户编写可以和 MATLAB 进行交互的 C 或 C++语言程序。另外，MATLAB 网页服务程序还容许在 Web 应用中使用自己的 MATLAB 数学和图形程序。

MATLAB 的一个重要特色就是它具有一套程序扩展系统和一组称之为工具箱的特殊应用子程序。工具箱是 MATLAB 函数的子程序库，每一个工具箱都是为某一类学科专业和应用而定制的，主要包括信号处理、控制系统、神经网络、模糊逻辑、小波分析和系统仿真等方面的应用。

1.3 MATLAB 系统的构成

MATLAB 系统由 MATLAB 开发环境、MATLAB 数学函数库、MATLAB 语言、MATLAB 图形处理系统和 MATLAB 应用程序接口（API）五大部分构成。

◆ MATLAB 开发环境

MATLAB 开发环境是一套方便用户使用 MATLAB 函数和文件的工具集，其中许多工具是图形化用户接口。它是一个集成化的工作空间，可以让用户输入、输出数据，并提供了 M 文件的集成编译和调试环境。它包括 MATLAB 桌面、命令窗口、M 文件编辑调试器、MATLAB 工作空间和在线帮助文档。

◆ MATLAB 数学函数库

MATLAB 数学函数库包括了大量的计算算法，从基本运算（如加法、正弦函数等）到复杂算法，如矩阵求逆、贝塞尔函数、快速傅立叶变换等。

◆ MATLAB 语言

MATLAB 语言是一个高级的基于矩阵/数组的语言，它有程序流控制、函数、数据结构、输入/输出和面向对象编程等特色。用户既可以用它来快速编写简单的程序，也可以用它来编写庞大复杂的应用程序。

◆ MATLAB 图形处理系统

图形处理系统使得 MATLAB 能方便地图形化显示向量和矩阵，而且能对图形添加标注和打印。它包括强力的二维及三维图形函数、图像处理和动画显示等函数。

◆ MATLAB 应用程序接口（API）

MATLAB 应用程序接口（API）是一个使 MATLAB 语言能与 C、Fortran 等其他高级编程语言进行交互的函数库，该函数库的函数通过调用动态链接库（DLL）实现与 MATLAB 文件的数据交换，其主要功能包括在 MATLAB 中调用 C 和 Fortran 程序，以及在 MATLAB 与其他应用程序间建立客户/服务器关系。

1.4 MATLAB 的工具箱

工具箱（Toolbox）是 MATLAB 的关键部分，它是 MATLAB 强大功能得以实现的载体和手段，它是对 MATLAB 基本功能的重要扩充。MATLAB 每年都会增加一些新的工具箱，所以，在一般情况下，工具箱的列表不是固定不变的，有关 MATLAB 工具箱的最新信息可以在 http://www.mathworks.com/products 中看到。

当前流行的 MATLAB 版本包括三十几个工具箱，工具箱又可以分为功能性工具箱和学科工具箱。功能性工具箱用来扩充 MATLAB 的符号计算、可视化建模仿真、文字处理以及与硬件实时交互等功能，能用于多种学科；学科工具箱是专业性比较强的工具箱，控制工具箱、信号处理工具箱、通信工具箱等都属于此类。

下面，将科学计算中常用的工具箱内所包含的主要内容做一下简要介绍。

1. 符号数学工具箱（Symbolic Math Toolbox）

- 符号表达式和符号矩阵的创建
- 符号微积分、线性代数、方程求解
- 因式分解、展开和简化
- 符号函数的二维图形
- 图形化函数计算器

2. 样条工具箱（Spline Toolbox）

- 分段多项式和 B 样条
- 样条的构造
- 曲线拟合及平滑
- 函数微积分

3. 最优工具箱（Optimization Toolbox）

- 线性规划和二次规划
- 求函数的最大值和最小位
- 多目标优化
- 约束条件下的优化
- 非线性方程求解

4. 偏微分方程工具箱（Partial Differential Equation Toolbox）

- 二维偏微分方程的图形处理
- 几何表示
- 自适应曲面绘制
- 有限元方法

5. 统计工具箱（Statistics Toolbox）

- 概率分布和随机数生成
- 多变量分析
- 回归分析
- 主元分析
- 假设检验

1.5 MATLAB 桌面操作环境

MATLAB 为用户提供了全新的桌面操作环境，了解并熟悉这些桌面操作环境是使用 MATLAB 的基础，下面介绍 MATLAB 的启动、主要功能菜单、命令窗口、工作空间、文件管理和帮助管理等。

1.5.1 MATLAB 启动和退出

以 Windows 操作系统为例，进入 Windows 后，选择“开始”→“程序”→“Matlab 7.0”，便可以进入如图 1-1 所示的 MATLAB 主窗口。如果安装时选择在桌面上生成快捷方式，也可以双击快捷方式直接启动。

在启动 MATLAB 且命令编辑区显示帮助信息后，将显示符号“|”，符号“|”表示 MATLAB 已准备好，正等待用户输入命令，这时就可以在提示符“|”后面键入命令，按下回车键后，MATLAB 就会解释执行所输入的命令，并在命令后面给出计算结果。如果在输入命令后以分号结束，按回车键后则不会显示结果。

退出 MATLAB 系统的方式有两种：

（1）在文件菜单（File）中选择“Exit”或“Quit”；

（2）用鼠标单击窗口右上角的关闭图标。

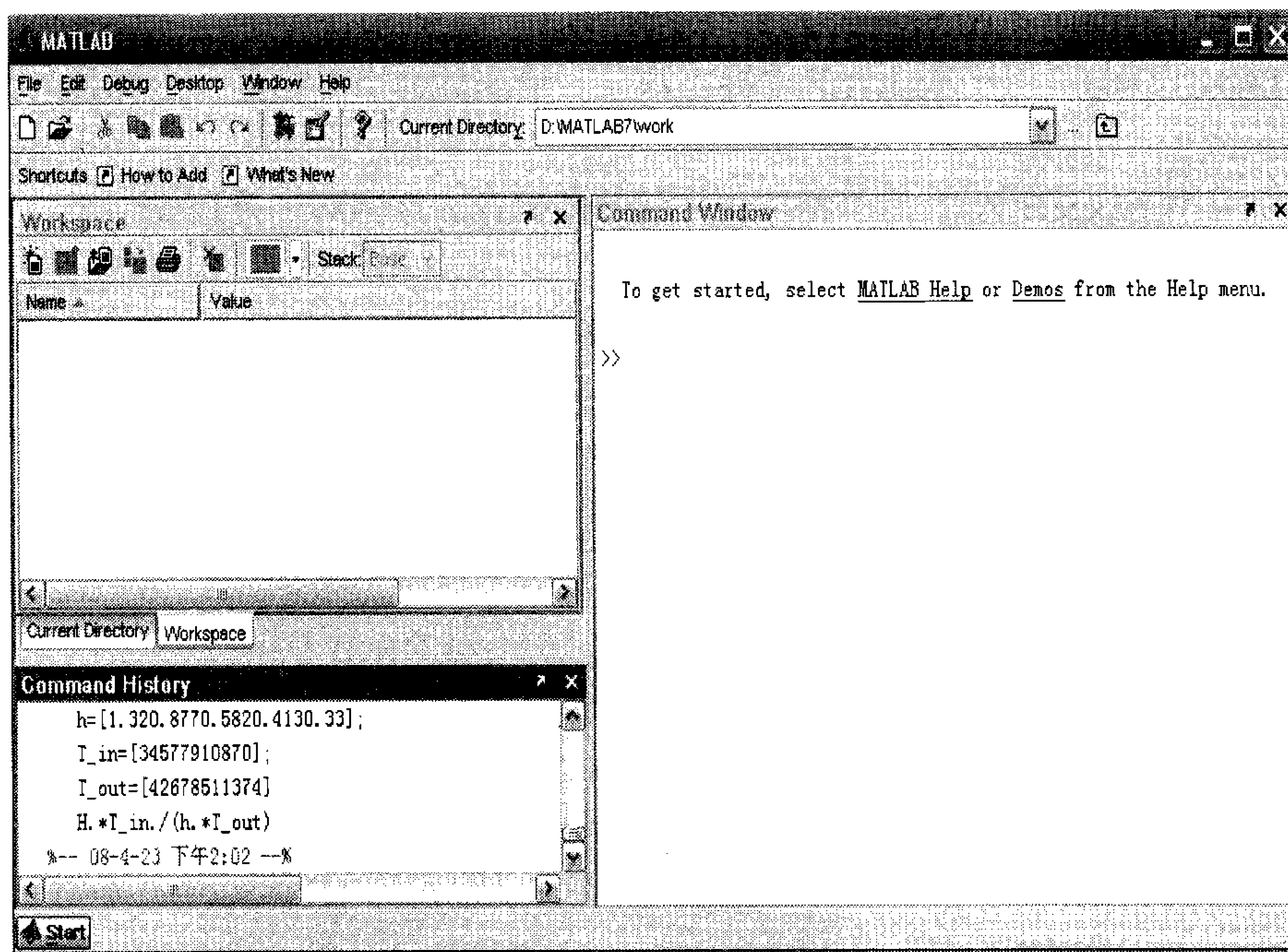


图 1-1　MATLAB 主窗口

1.5.2　MATLAB 主菜单及功能

打开 MATLAB 主窗口后，即弹出其主菜单栏，主菜单栏的各菜单项及其下拉菜单的功能如下所述。

1．File 主菜单项

单击 File 主菜单项或同时按下“Alt+F”组合键，弹出如图 1-2 所示的 File 下拉菜单。其中，带下划线的字母表示快捷键，即单击该字母键也可执行相应的功能。

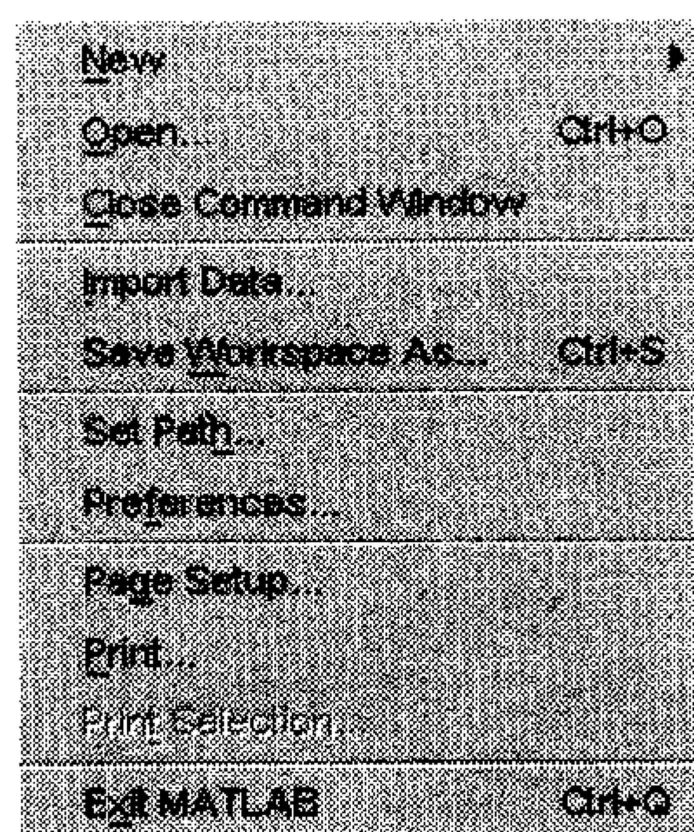


图 1-2　File 下拉菜单

（1）New：用于建立新的.m 文件、图形、模型和图形用户界面。

（2）Open：用于打开 MATLAB 的.m 文件、.fig 文件、.mat 文件、.mdl 文件、.cdr 文件等，也可通过快捷键“Ctrl+O”来实现此项操作。

（3）Close Command Window：关闭命令窗口。

（4）Import Data：用于从其他文件导入数据，单击后弹出对话框，选择导入文件的路径和位置。

（5）Save Workspace As：用于把工作空间的数据存放到相应的路径文件中。

（6）Set Path：设置工作路径。

（7）Preferences：用于设置命令窗的属性，单击该选项弹出如图 1-3 所示的属性画面。

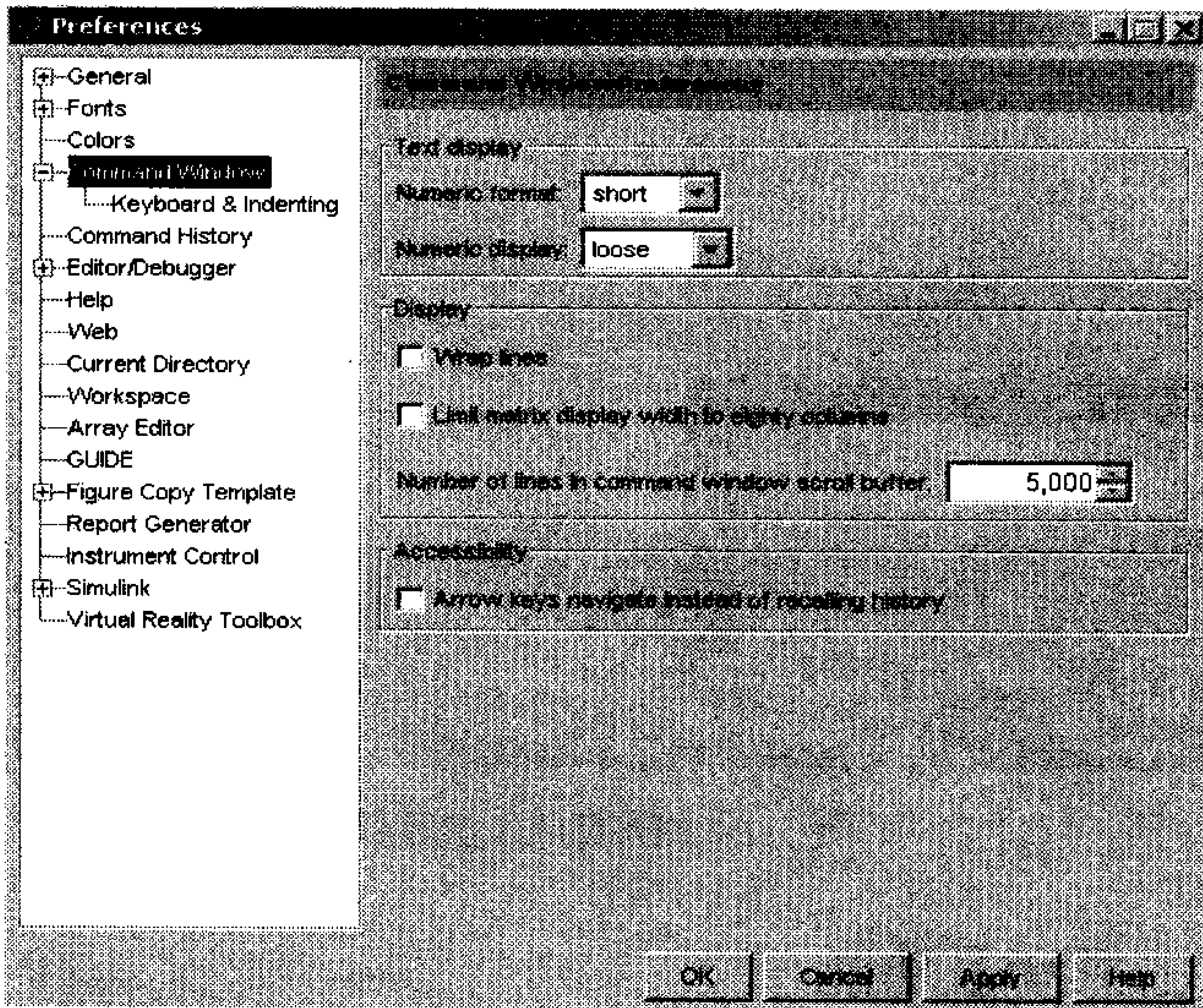


图 1-3　命令窗口属性对话框

（8）Page Setup：用于页面设置。

（9）Print：用于设置打印属性。

（10）Print Selection：用于对选择的文件数据进行打印设置。

（11）Exit MATLAB：退出 MATLAB 桌面操作环境。

2．Edit 主菜单项

单击 Edit 主菜单项或同时按下“Alt+E”组合键，弹出如图 1-4 所示的下拉菜单。

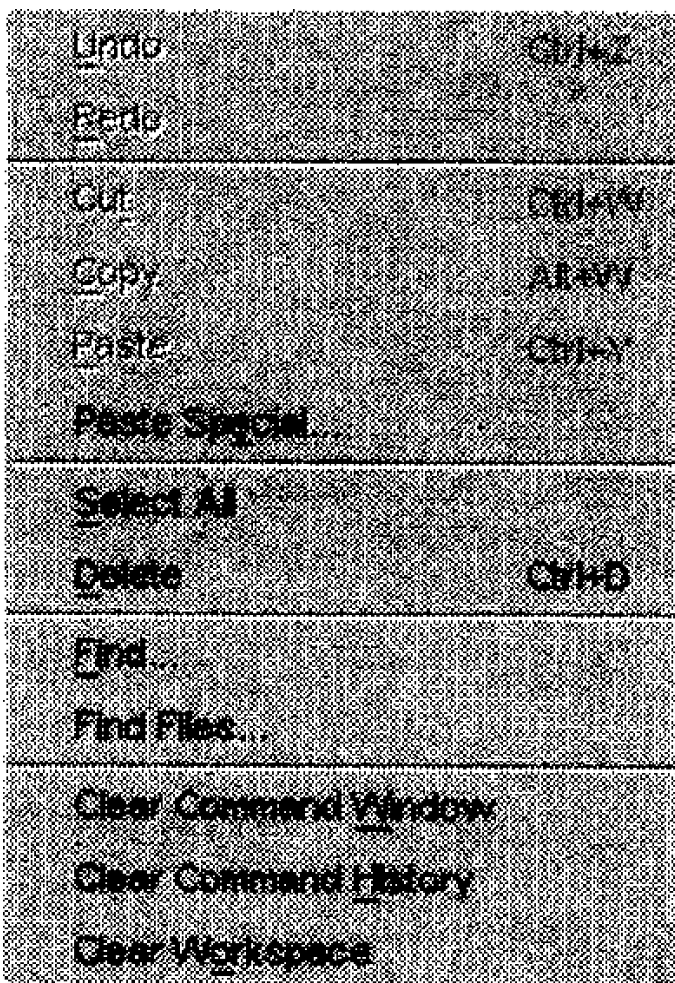


图 1-4　Edit 下拉菜单

（1）Undo：用于撤销上一步操作，也可通过快捷键“Ctrl+Z”来实现此项操作。
（2）Redo：用于重新执行上一步操作。
（3）Cut：用于剪切选中的对象，也可通过快捷键“Ctrl+W”来实现此项操作。
（4）Copy：用于复制选中的对象，也可通过快捷键“Alt+W”来实现此项操作。
（5）Paste：用于粘贴剪贴板上的内容，也可通过快捷键“Ctrl+Y”来实现此项操作。
（6）Paste Special：用于特定内容的粘贴。
（7）Select All：用于全部选择。
（8）Delete：用于删除所选的对象，也可通过快捷键“Ctrl+D”来实现此项操作。
（9）Find：用于查找所需选择的对象。
（10）Find Files：用于查找所需文件。
（11）Clear Command Window：用于清除命令窗口区的对象。
（12）Clear Command History：用于清除命令窗口区的历史记录。
（13）Clear Workspace：用于清除工作区的对象。

3．Debug 主菜单项

单击 Debug 主菜单项或同时按下“Alt+B”组合键，弹出如图 1-5 所示的下拉菜单。

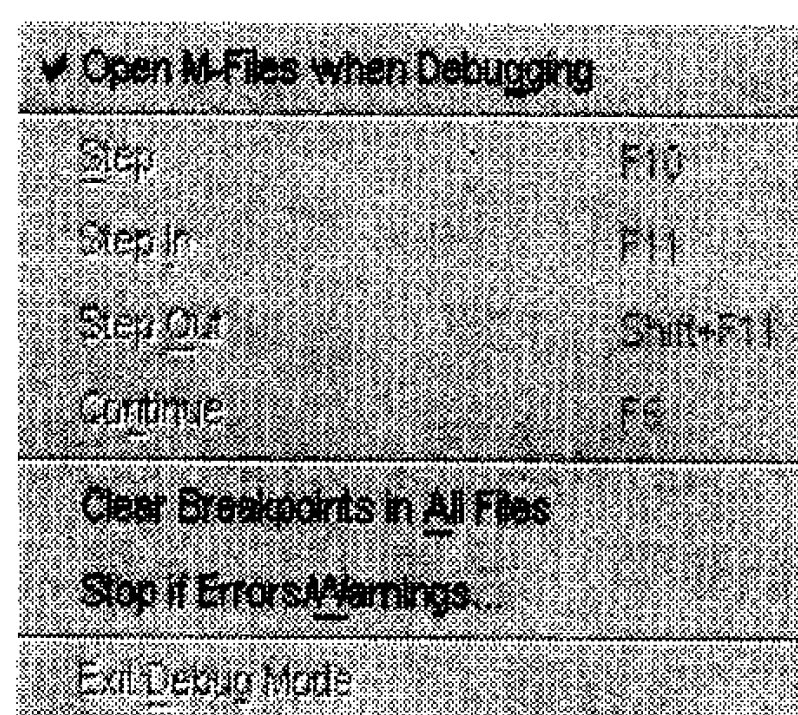


图 1-5 Debug 下拉菜单

（1）Open M-Files when Debugging：用于调试时打开 M 文件。
（2）Step：用于单步调试程序，也可通过快捷键“F10”来实现此项操作。
（3）Step In：用于单步调试进入子函数，也可通过快捷键“F11”来实现此项操作。
（4）Step Out：用于单步调试从子函数中跳出，也可通过快捷键“Shift+F11”来实现此项操作。
（5）Continue：程序执行到下一断点，也可通过快捷键“F5”来实现此项操作。
（6）Clear Breakpoints in All Files：清除所有打开文件中的断点。
（7）Stop if Errors/Warnings：在程序出错或报警处停止往下执行。
（8）Exit Debug Mode：退出调试模式。

4．Desktop 主菜单项

单击 Desktop 主菜单项或同时按下“Alt+D”组合键，弹出如图 1-6 所示的下拉菜单。
（1）Undock Command Window：将命令窗口变为全屏显示，并设为当前活动窗口。
（2）Desktop Layout：单击该项后，弹出如图 1-7 所示的子菜单；用于工作区的设置，

其设置选项包括系统默认设置项（Default）、单独命令窗口项（Command Window Only）、命令历史窗口和命令窗口项（History and Command Window）、全部标签项显示（All Tabbed）。

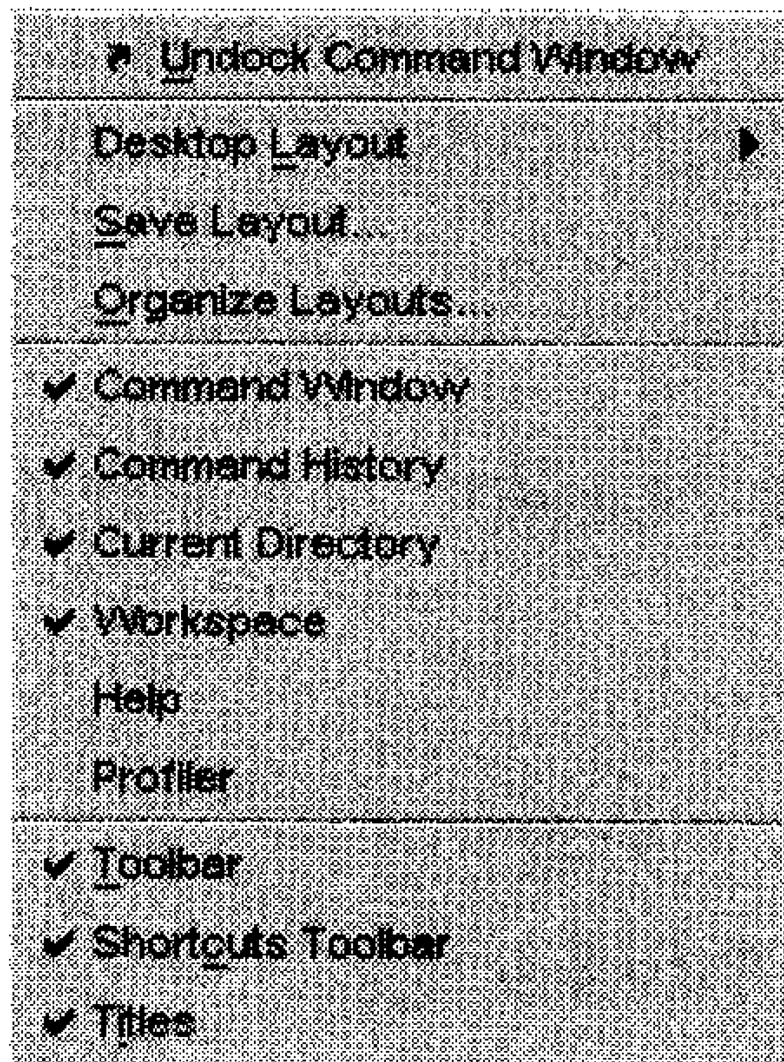


图 1-6　Desktop 下拉菜单

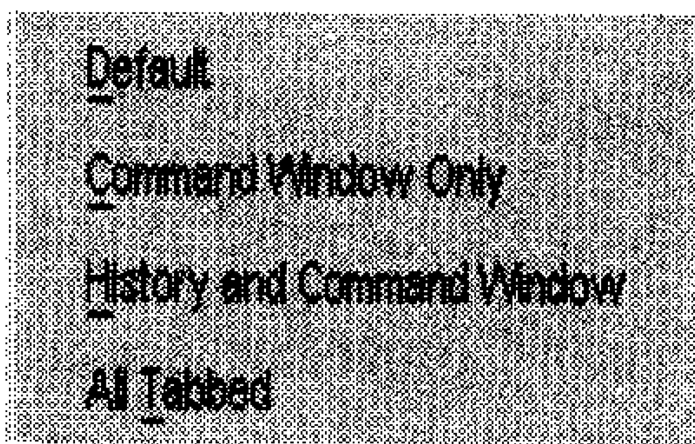


图 1-7　Desktop Layout 弹出子菜单

（3）Save Layout：保存选定的工作区设置。

（4）Organize Layouts：管理保存的工作区设置。

（5）Command Window：命令窗口项，选择该项，屏幕上便会显示相应窗口。

（6）Command History：命令历史窗口项，选择该项，屏幕上便会显示相应窗口。

（7）Current Directory：当前路径窗口项，选择该项，屏幕上便会显示相应窗口。

（8）Workspace：工作窗口项，选择该项，屏幕上便会显示相应窗口。

（9）Help：帮助窗口项，选择该项，屏幕上便会显示相应窗口。

（10）Profiler：轮廓图窗口项，选择该项，屏幕上便会显示相应窗口。

（11）Toolbar：显示或隐藏工具栏选项。

（12）Shortcuts Toolbar：显示或隐藏快捷方式选项。

（13）Titles：显示或隐藏标题栏选项。

5. Window 主菜单项

单击 Window 主菜单项或同时按下“Alt+W”组合键，弹出如图 1-8 所示的下拉菜单。

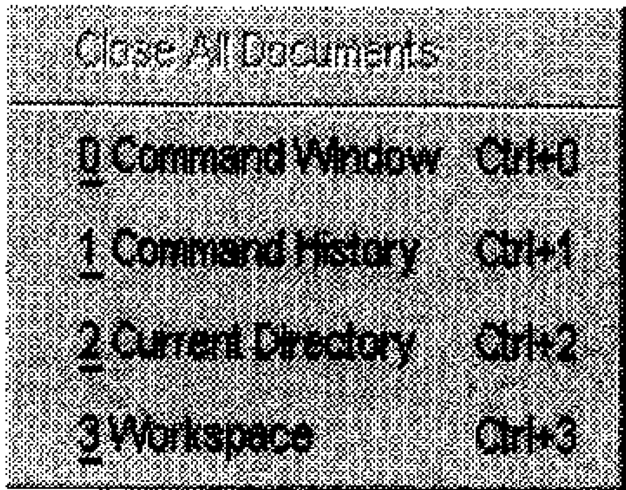


图 1-8　Window 下拉菜单

（1）Close All Documents：关闭所有文档。

（2）0 Command Window：选定命令窗口为当前活动窗口，也可通过快捷键“Ctrl+0”来实现此项操作。

（3）1 Command History：选定命令历史窗口为当前活动窗口，也可通过快捷键“Ctrl+1”来实现此项操作。

（4）2 Current Directory：选定当前路径窗口为当前活动窗口，也可通过快捷键“Ctrl+2”来实现此项操作。

（5）3 Workspace：选定工作空间窗口为当前活动窗口，也可通过快捷键“Ctrl+3”来实现此项操作。

6. Help 主菜单项

单击 Help 主菜单项或同时按下“Alt+H”组合键，弹出如图 1-9 所示的下拉菜单。

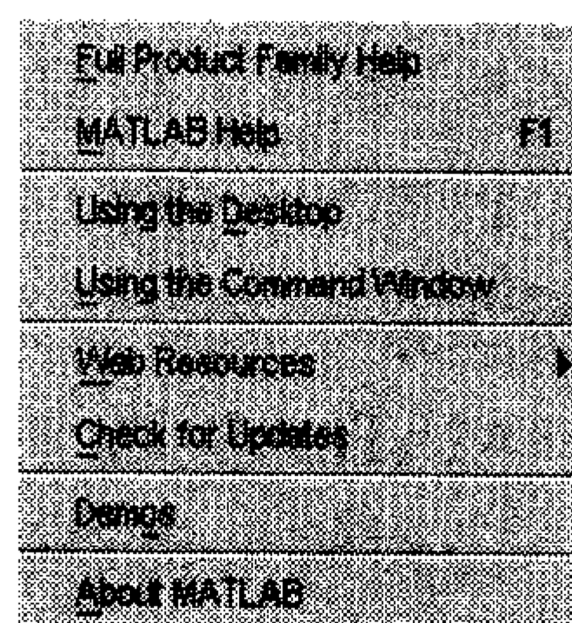


图 1-9　Help 下拉菜单

（1）Full Product Family Help：显示所有 MATLAB 产品的帮助信息。

（2）MATLAB Help：启动 MATLAB 帮助。

（3）Using the Desktop：启动 Desktop 的帮助。

（4）Using the Command Window：启动命令窗口的帮助。

（5）Web Resources：显示 Internet 上一些相关的资源网址。

（6）Check for Updates：检查软件是否更新。

（7）Demos：调用 MATLAB 所提供的范例程序。

（8）About MATLAB：显示有关 MATLAB 的信息。

1.5.3　MATLAB 命令窗口

MATLAB 的命令窗口如图 1-10 所示，它用于 MATLAB 命令的交互操作，具有两大主要功能：

（1）提供用户输入命令的操作平台，用户通过该窗口输入命令和数据；

（2）提供命令执行结果的显示平台，该窗口显示命令执行的结果。

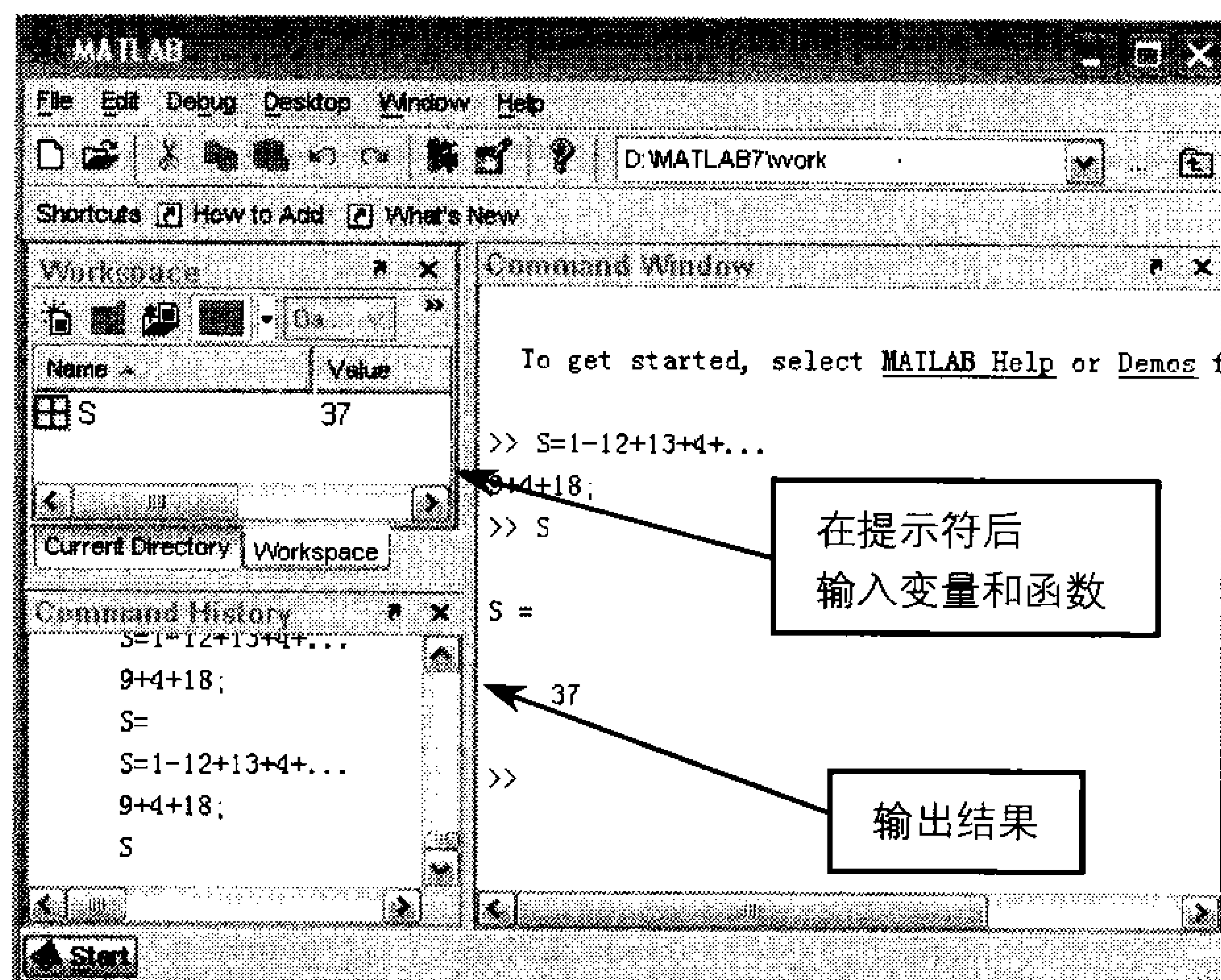


图 1-10　MATLAB 的命令窗口

在命令窗口内执行的 MATLAB 主要操作有：

- 运行函数和输入变量；
- 控制输入和输出；
- 执行程序，包括 M 文件和外部程序；
- 保存一段日志；
- 打开或关闭其他应用窗口；
- 各应用窗口的参数选择。

计算机安装好 MATLAB 之后，双击 MATLAB 图标，就可以进入命令窗口，此时意味着系统处于准备接受命令的状态，可以在命令窗口中直接输入命令语句。

MATLAB 语句形式为：>>变量 = 表达式。

通过等号将表达式的值赋予变量。当键入回车键时，该语句被执行。语句执行之后，窗口自动显示出语句执行的结果。

使用方向键和控制键可以编辑、修改已输入的命令，↑键回调上一行命令，↓键回调下一行命令。使用“more off”表示不允许分页，“more on”表示允许分页，“more (n)”表示指定每页输出的行数。回车前进一行，空格键显示下一页，“q”结束当前显示。

如果命令语句超过一行或者太长希望分行输入，则可以使用多行命令继续输入。例如，输入下列式子时，可以通过两行输入。

```
>> S=1-12+13+4+...
9+4+18;
>> S
S =    37
```

说明 三个小黑点是“连行号”，分号“；”作用是：指令执行结果将不显示在屏幕上，但变量 S 将驻留在内存中。

MATLAB 提供了一组可以在命令窗口中输入的命令，以执行相应的操作，常用的命令及功能如表 1.1 中所示。

表 1.1　命令窗口中常用的命令及功能

命　令	功　能
clc	擦去一页命令窗口，光标回屏幕左上角
clear	清除工作空间中所有的变量
clear all	从工作空间清除所有变量和函数
clear　变量名	清除指定的变量
clf	清除图形窗口内容
delete <文件名>	从磁盘中删除指定文件
help <命令名>	查询所列命令的帮助信息
which <文件名>	查找指定文件的路径
who	显示当前工作空间中所有变量的一个简单列表
whos	列出变量的大小、数据格式等详细信息

续表

命 令	功 能
what	列出当前目录下的.m 文件和.mat 文件
load name	下载'name'文件中的所有变量到工作空间
load name x y	下载'name'文件中的变量 x,y 到工作空间
save name	保存工作空间变量到文件 name.mat 中
save name x y	保存工作空间变量 x、 y 到文件 name.mat 中
pack	整理工作空间内存
size(变量名)	显示当前工作空间中变量的尺寸
length(变量名)	显示当前工作空间中变量的长度
disp(变量名)	显示当前工作空间中变量
“↑” 或 “Ctrl+P”	调用上一行的命令
“↓” 或 “Ctrl+N”	调用下一行的命令
“←” 或 “Ctrl+B”	退后一格
“→” 或 “Ctrl+F”	前移一格
Home 或 “Ctrl+A”	光标移到行首
End 或 “Ctrl+E”	光标移到行尾
Esc 或 “Ctrl+U”	清除一行
Del 或 “Ctrl+D”	清除光标后字符
Backspace 或 “Ctrl+H”	清除光标前字符
“Ctrl+K”	清除光标至行尾字
“Ctrl+C”	中断程序运行

1.5.4 MATLAB 工作空间

MATLAB 的工作空间如图 1-11 所示。

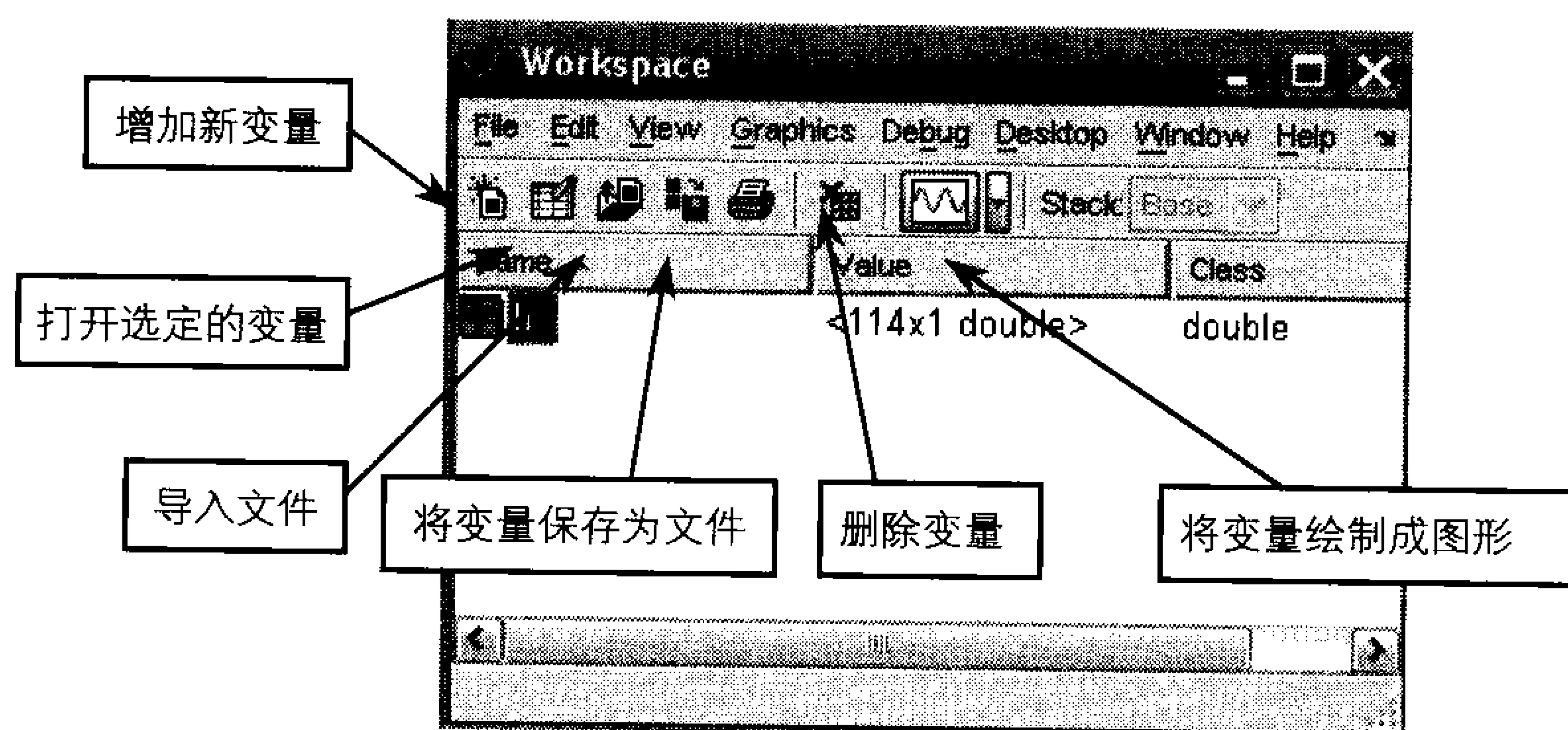


图 1-11 MATLAB 的工作空间

工作空间中的变量以变量名（Name）、数值（Value）和类型（Class）的形式显示出来，双击某个变量，将进入矩阵编辑器（Array Editor），可以直接观察变量中具体元素的值，也可以直接修改这些元素。

1.5.4.1 工作空间的工具条

MATLAB 7 的工作空间中还有一个工具条，可快捷地在工作空间中进行许多操作，这些操作在图 1-11 中标注出来了，简单介绍如下。

- （增加新变量）：在工作空间中增加一个新的变量，并可对此变量进行赋值、修改等操作。
- （打开选定的变量）：将工作空间中选定的变量在矩阵编辑器（Array Editor）中打开，可对此变量进行修改等操作。
- （导入文件）：将 MATLAB 支持格式的文件导入到工作空间中。
- （将变量保存为文件）：将工作空间中选定的变量以文件的形式保存起来。
- （删除变量）：将工作空间中选定的变量删除。
- （将变量绘制成图形）：将工作空间中选定的变量绘制成图形，支持的绘图函数有 plot、bar、stem、stairs、area、pie、hist 和 plot3 等。

1.5.4.2 工作空间相关的常用命令

MATLAB 还有几个常用的工作空间操作的命令，分别是 who、whos、clear、size、length，其各自功能描述如下。

- who：显示当前工作空间中所有变量的一个简单列表。
- whos：列出变量的大小、数据格式等详细信息。
- clear：清除工作空间中的所有变量。
- clear 变量名：清除指定的变量。
- size(a)：获取向量 a 的行数与列数。
- length(a)：获取向量 a 的长度，并在屏幕上显示。如果 a 是矩阵，则显示的参数为行数中的最大值。

1.5.4.3 工作空间的数据存取函数

MATLAB 提供了以下保存（save）和载入（load）工作空间的函数。

1. save 函数

save 命令是将 MATLAB 工作空间中的变量存入磁盘，具体格式介绍如下。

- save：将当前 MATLAB 工作空间中所有变量以二进制格式存入名为 matlab.mat（默认的文件名）的文件中。
- save dfile（文件名）：将当前工作空间中所有变量以二进制格式存入名为 dfile.mat 文件，扩展名自动产生。
- save dfile x：只把变量 x 以二进制格式存入 dfile.mat 文件，扩展名自动产生。
- save dfile.dat x –ascii：将变量 x 以 8 位 ASCII 码形式存入 dfile.mat 文件。
- save dfile.dat x –ascii –double：将变量 x 以 16 位 ASCII 码形式存入 dfile.mat 文件。
- save（fname，'x'，'-ascii'）：fname 是一个预先定义好的包含文件名的字符串，

该用法将变量 x 以 ASCII 码形式存入由 fname 定义的文件中，由于在这种用法中，文件名是一个字符变量，因此可以方便地通过编程的方法存储一系列数据文件。

2. load 函数

load 命令是将磁盘上的数据读入到工作空间，具体格式介绍如下。

- load：把磁盘文件 matlab.mat（默认的文件名）的内容读入内存，由于存储.mat 文件时已包含了变量名的信息，因此调回时已直接将原变量信息带入，不需要重新赋值变量。
- load dfile：把磁盘文件 dfile.mat 的内容读入内存。
- load dfile.dat：把磁盘文件 dfile.mat 的内容读入内存，这是一个 ASCII 码文件，系统自动将文件名（dfile）定义为变量名。
- x=load (fname)：fname 是一个预先定义好的包含文件名的字符串，将由 fname 定义文件名的数据文件读入变量 x 中，使用这种方法可以通过编程方便地调入一系列数据文件。

1.5.5 MATLAB 文件管理

MATLAB 提供了一组文件管理命令，包括列文件名、显示或删除文件、显示或改变当前目录等，相关的命令及其功能如表 1.2 所示。

表 1.2 MATLAB 常用文件管理命令

命　令	功　能	命　令	功　能
what	显示当前目录与 MATLAB 相关的文件及路径	type filename	在命令窗口中显示文件 filename
dir	显示当前目录下所有的文件	delete filename	删除文件 filename
which	显示某个文件的路径	cd ..	返回上一级目录
cd path	由当前目录进入 path 目录	cd	显示当前目录

1.5.6 MATLAB 帮助使用

MATLAB 的所有函数都是以逻辑群组的方式进行组织的，而 MATLAB 的目录结构就是以这些群组方式来编排的，几个常用的帮助命令介绍如下。

（1）helpwin：帮助窗口。

（2）helpdesk：帮助桌面，浏览器模式。

（3）lookfor：返回包含指定关键词的项。

（4）demo：打开示例窗口。

MATLAB 还提供了丰富的 help 命令，如表 1.3 所示，在命令窗口中输入相关命令就可以获得相关的帮助。

表 1.3　MATLAB 常用帮助命令

命　令	功　能	命　令	功　能
help matfun	矩阵函数 – 数值线性代数	help datafun	数据分析和傅立叶变换函数
help general	通用命令	help ops	操作符和特殊字符
help graphics	通用图形函数	help polyfun	多项式和内插函数
help elfun	基本的数学函数	help lang	语言结构和调试
help elmat	基本矩阵和矩阵操作	help strfun	字符串函数
help control	控制系统工具箱函数		

1.6 小结

本章首先概要讲述了 MATLAB 语言的产生和发展历程、其优势及特点，然后一一讲述了 MATLAB 的系统结构、工具箱和桌面操作环境，本章是全书内容的基础，需要扎实掌握。

第 2 章　MATLAB 计算基础

MATLAB 是一门计算语言，它的运算指令和语法基于一系列基本的矩阵运算以及它们的扩展运算，它还支持复数这一数值元素，这也是 MATLAB 区别于其他高级语言的最大特点之一，它给许多领域的计算带来了极大方便。

2.1 MATLAB 数值类型

MATLAB 包括 4 种基本数据类型，即双精度数组、字符串数组、元胞数组、构架数组。数值之间可以相互转化，这为其计算功能开拓了广阔的空间。

1. 变量与常量

变量是数值计算的基本单元。与 C 语言等其他高级语言不同，MATLAB 语言中的变量无须事先定义，一个变量以其名称在语句命令中第一次合法出现而定义，运算表达式变量中不允许有未定义的变量，也不需要预先定义变量的类型，MATLAB 会自动生成变量，并根据变量的操作确定其类型。

（1）MATLAB *变量命名规则*

MATLAB 中的变量命名规则如下：

1）变量名区分大小写，因此 A 与 a 表示的是不同的变量；

2）变量名以英文字母开始，第一个字母后可以使用字母、数字、下划线，但不能使用空格和标点符号；

3）变量名长度不得超过 31 位，超过的部分将被忽略；

4）某些常量也可以作为变量使用，如 i 在 MATLAB 中表示虚数单位，但也可以作为变量使用。

常量是指那些在 MATLAB 中已预先定义其数值的变量，默认的常量如表 2.1 所示。

表 2.1　MATLAB 默认常量

名　称	说　明	名　称	说　明
Pi	圆周率	eps	浮点数的相对误差
INF（或 inf）	无穷大	i（或 j）	虚数单位，定义为 $\sqrt{-1}$
NaN（或 nan）	代表不定值（即 0/0）	nargin	函数实际输入参数个数
realman	最大的正实数	nargout	函数实际输出参数个数
realmix	最小的正实数	ANS（或 ans）	默认变量名，以应答最近一次操作运算结果

（2）MATLAB 变量的显示

任何 MATLAB 语句的执行结果都可以在屏幕上显示，同时赋值给指定的变量，没有指定变量时，赋值给一个特殊的变量 ans，数据的显示格式由 format 命令控制。format 只影响结果的显示，不影响其计算与存储。MATLAB 总是以双字长浮点数（双精度）来执行所有的运算。如果结果为整数，则显示没有小数；如果结果不是整数，则输出形式有表 2.2 所示的几种形式。

表 2.2　MATLAB 的数据显示格式

格　式	含　义	格　式	含　义
format （short）	短格式（5 位定点数）	format long e	长格式 e 方式
format long	长格式（15 位定点数）	format bank	2 位十进制格式
format short e	短格式 e 方式	format hex	十六进制格式

（3）MATLAB 变量的存取

工作空间中的变量可以用 save 命令存储到磁盘文件中。键入命令“save<文件名>”，将工作空间中全部变量存到“<文件名>.mat”文件中去，若省略“<文件名>”则存入文件“matlab.mat”中；命令“save<文件名><变量名集>”将“<变量名集>”指出的变量存入文件“<文件名>.mat”中。

用 load 命令可将变量从磁盘文件读入 MATLAB 的工作空间，其用法为“load<文件名>”，它将“<文件名>”指出的磁盘文件中的数据依次读入名称与“<文件名>”相同的工作空间中的变量中去。若省略“<文件名>”则“matlab.mat”从中读入所有数据。

用 clear 命令可从工作空间中清除现存的变量。

2．字符串

字符是 MATLAB 中符号运算的基本元素，也是文字等表达方式的基本元素，在 MATLAB 中，字符串作为字符数组用单引号（'）引用到程序中，还可以通过字符串运算组成复杂的字符串。字符串数值和数字数值之间可以进行转换，也可以执行字符串的有关操作。

3．元胞数组

元胞是元胞数组（Cell Array）的基本组成部分。元胞数组与数字数组相似，以下标来区分，单元胞数组由元胞和元胞内容两部分组成。用花括号{ }表示元胞数组的内容，用圆括号()表示元胞元素。与一般的数字数组不同，元胞可以存放任何类型、任何大小的数组，而且同一个元胞数组中各元胞的内容可以不同。

例 2-1　元胞数组创建与显示实例。

解：MATLAB 程序代码如下。

```
    A(1, 1)={'An example of cell array'};
    A(1, 2)={[1 2;3 4]}; A{2, 1}=tf (1, [1, 8]); A{2, 2}={A(1, 2);'This is an
example'};
    celldisp(A)  %显示该元胞数组
```

元胞数组 A 第 1 行用元胞数组标志法建立一个字符串和一个矩阵；第 2 行用元胞内容编址法，建立一个传递函数和一个由两个元素组成的元胞组，该元胞组分别是矩阵和字符串，最后，用 celldisp 函数显示该元胞数组 A。

4. 构架数组

与元胞数组相似，构架数组（Structure Array）也能存放各类数据，使用指针方式传递数值。构架数组由结构变量名和属性名组成，用指针操作符"."连接结构变量名和属性名。例如，可用 parameter.temperature 表示某一对象的温度参数，用 parameter.humidity 表示某一对象的湿度参数等，因此，该构架数组 parameter 由两个属性组成。

5. 对象

面向对象的 MATLAB 语言采用了多种对象，如自动控制中常用的传递函数模型对象（tf object）、状态空间模型对象（ss object）和零极点模型对象（zpk object），一些对象之间可以相互转换，例如可以从传递函数模型对象转化为零极点模型对象，这将在后面具体介绍。

2.2 关系运算和逻辑运算

除了传统的数学运算外，MATLAB 还支持关系运算和逻辑运算。如果你已经有了一些编程经验，那对这些运算不会陌生。这些操作符和函数的目的是提供求解真/假命题的答案。关系运算和逻辑运算主要用于控制基于真/假命题的各 MATLAB 命令（通常在 M 文件中）的流程或执行次序。

作为所有关系表达式和逻辑表达式的输入，MATLAB 把任何非 0 数值当做真，把 0 当做假。所有关系表达式和逻辑表达式的输出，对于真输出为 1，对于假输出为 0。

MATLAB 为关系运算和逻辑运算提供了关系操作符和逻辑操作符，如表 2.3 和表 2.4 所示。

表 2.3 关系运算符符号

符　号	功　能	符　号	功　能
<	小于	>=	大于等于
<=	小于等于	==	等于
>	大于	~=	不等于

表 2.4 逻辑运算符

符　号	功　能	符　号	功　能
&	逻辑与	~	逻辑非
|	逻辑或		

此外，MATLAB 还提供了若干关系运算函数和逻辑运算函数，分别如表 2.5 和表 2.6 所示。

表 2.5 关系运算函数

函 数 名	功 能	函 数 名	功 能
all	所有向量为非零元素时为真	xor	逻辑异或运算
any	任一向量为非零元素时为真		

表 2.6 逻辑运算函数

函 数 名	功 能	函 数 名	功 能
Bitand	位方式的逻辑与运算	Bitcmp	位比较运算
Bitor	位方式的逻辑或运算	Bitmax	最大无符号浮点整数
Bitxor	位方式的逻辑异或运算	Bitshift	将二进制移位运算

2.3 矩阵及其运算

MATLAB 软件的最大特色是强大的矩阵计算功能，在 MATLAB 软件中，所有的计算都是以矩阵为单元进行的，可见矩阵是 MATLAB 的核心。下面以表格的形式列出 MATLAB 提供的每类矩阵运算的函数，并各举一个实例进行说明，同类函数的用法基本类似，详细的用法及函数内容说明可参考联机帮助。

2.3.1 矩阵的创建

由 m 行 n 列构成的数组 a 称为 $m\times n$ 阶矩阵，它总共由 $m\times n$ 个元素组成，矩阵元素记为 a_{ij}，其中 i 表示行，j 表示列。

当 $m=n$ 时，矩阵 a 称为方阵。当 $i\neq j$ 时，所有的 $a_{ij}=0$，且 $m=n$，得到的矩阵称为对角阵。

当对角阵的对角线上的元素全为 1 时，称为单位阵，记为 $\boldsymbol{I}$。

对于 $(m\times n)$ 阶矩阵 w，当 $w_{ij}=a_{ji}$ 时，称 w 是 a 的转置矩阵，记为 $w=a'$。

对于 a 为 $(m\times 1)$ 的形式时，称 a 是 m 个元素的列向量，对于 a 为 $(1\times n)$ 的形式时，称 a 是 n 个元素的行向量。

矩阵的表现形式和数组相似，它以左方括号“[”开始，以右方括号“]”结束，每一行元素结束用行结束符号（分号“;”）或回车符分割，每个元素之间用元素分割符号（空格或“,”）分隔。建立矩阵的方法有直接输入矩阵元素、在现有矩阵中添加或删除元素、读取数据文件、采用现有矩阵组合、矩阵转向、矩阵移位及直接建立特殊矩阵等。

例 2-2 矩阵创建实例。

解： MATLAB 程序代码如下。

```
>> a=[1 2 3;4 5 6]
```

运行结果是创建了一个 2×3 的矩阵 a，a 的第 1 行由 1、2、3 这 3 个元素组成，第 2 行由 4、5、6 这 3 个元素组成，输出结果如下：

```
a = 1    2    3
    4    5    6
```

接着输入：

```
>> b=[a; 11, 12, 13]  %添加一行元素[11, 12, 13]
```

运行结果是创建了一个 3×3 的矩阵 b，b 矩阵是在 a 矩阵的基础上添加一行元素 11、12、13，组成一个 3×3 矩阵，输出结果如下：

```
b = 1    2    3
   4    5    6
   11   12   13
```

MATLAB 中对矩阵元素的访问如下所示：

- 单个元素的访问：b(3, 2)→12，访问了第 3 行和第 2 列交叉的元素；
- 整列元素的访问：b(:, 3)→[3, 6, 13]'，访问了第 3 列中的所有元素；
- 整行元素的访问：b(1, :)→[1, 4, 11]，访问了第 1 行中的所有元素；
- 整块元素的访问：b(2:3, 2:3)→[5, 6; 12, 13]，访问了一个（2×2）的子块矩阵。

MATLAB 提供了很多个特殊矩阵的生成函数，表 2.7 列出了一些常用的生成函数，关于其他的特殊矩阵生成函数及使用格式，请参见联机帮助。

表 2.7 MATLAB 常用特殊矩阵生成函数

函 数	功能说明	函 数	功能说明
zeros( )	生成元素全为 0 的矩阵	tril( )	生成下三角矩阵
ones( )	生成元素全为 1 的矩阵	eye( )	生成单位矩阵
rand( )	生成均匀分布随机矩阵	company( )	生成伴随矩阵
randn( )	生成正态分布随机矩阵	hilb( )	生成 Hilbert 矩阵
magic( )	生成魔方矩阵	vander( )	生成 vander 矩阵
diag( )	生成对角矩阵	hankel( )	生成 hankel 矩阵
triu( )	生成上三角矩阵	hadamard( )	生成 hadamard 矩阵

例 2-3 特殊矩阵生成函数使用实例。

解： MATLAB 程序代码如下。

```
>> a=[1, 2, 3; 4, 5, 6; 7, 8, 9];  b=tril(a)  %生成下三角矩阵
```

运行结果是生成了 b 矩阵，它是调用下三角矩阵生成函数 tril( )生成的 a 矩阵的下三角矩阵，输出结果如下：

```
b = 1    0    0
    4    5    0
    7    8    9
```

2.3.2 矩阵的运算

矩阵与矩阵之间可以进行如表 2.8 所示的基本运算。



在进行左除“/”和右除“\”时，两矩阵的维数必须相等。

表 2.8　矩阵基本运算

操作符号	功能说明	操作符号	功能说明
+	矩阵加法	/	矩阵的左除
−	矩阵减法	‘	矩阵转置
*	矩阵乘法	logm( )	矩阵对数运算
^	矩阵的幂	expm( )	矩阵指数运算
\	矩阵的右除	inv( )	矩阵求逆

例 2-4　矩阵基本运算实例。

解： MATLAB 程序代码如下。

```
>> a=[1, 2; 3, 4]; b=[3, 5; 2, 9]; div1=a/b; %矩阵的左除
>>div2=b\a  %矩阵的右除
```

两矩阵 a 与 b 进行了左除和右除运算，输出结果如下：

```
div1 =  0.2941    0.0588          div2= -0.3529   -0.1176
        1.1176   -0.1765                 0.4118    0.4706
```

MATLAB 提供了多种关于矩阵的函数，表 2.9 列出了一些常用的矩阵函数运算。

表 2.9　常用矩阵函数运算

函数名	功能说明	函数名	功能说明
rot90( )	矩阵逆时针旋转 90°	eig( )	计算矩阵的特征值和特征向量
flipud( )	矩阵上下翻转	rank( )	计算矩阵的秩
fliplr( )	矩阵左右翻转	trace( )	计算矩阵的迹
flipdim( )	矩阵的某维元素翻转	norm( )	计算矩阵的范数
shiftdim( )	矩阵的元素移位	poly( )	计算矩阵的特征方程的根

例 2-5　矩阵函数运算实例。

解： MATLAB 程序代码如下。

```
>> a=[1, 3, 5; 2, 4, 6; 7, 9, 13]; [b, c]=eig(a)  %求取矩阵的特征值和特征向量
```

通过函数 eig( )计算矩阵 a 的特征向量 b 和特征值 c，输出结果如下：

```
b=-0.3008   -0.7225    0.2284
  -0.3813   -0.3736   -0.8517
  -0.8742    0.5817    0.4717
c = 19.3341        0         0
         0   -1.4744         0
         0         0    0.1403
```

矩阵分解常用于方程求根，表 2.10 列出了一些常用的矩阵分解运算。

表 2.10 常用矩阵分解运算函数

函 数 名	功 能 说 明	函 数 名	功 能 说 明
eig( )	矩阵的特征值分解	svd( )	矩阵的奇异值分解
qr( )	矩阵的 QR 分解	chol( )	矩阵的 Cholesky 分解
schur( )	矩阵的 Schur 分解	lu( )	矩阵的 LU 分解

例 2-6 矩阵分解运算函数使用实例。

解：MATLAB 程序代码如下。

```
>> a=[6, 2, 1; 2, 3, 1; 1, 1, 1];  [L, U, P]=lu(a)  %对矩阵进行 LU 分解
```

通过函数 lu( )对矩阵 a 进行 LU 分解，得到上三角阵 U、下三角阵 L、置换矩阵 P，输出结果如下：

```
L =1.0000          0         0                U =  6.0000    2.0000    1.0000
   0.3333    1.0000         0                          0    2.3333    0.6667
   0.1667    0.2857    1.0000                          0         0    0.6429
P = 1     0     0
    0     1     0
    0     0     1
```

2.4 复数及其运算

复数以及在其基础上发展起来的复变函数这一重要的数学分支，解决了许多实变函数无法解决的问题，有着广泛的工程应用。

复变函数是控制工程的数学基础，常用来描述控制系统模型的传递函数，就是属于复变函数。MATLAB 支持在运算和函数中使用复数或复数矩阵，还支持复变函数运算。

2.4.1 复数的表示

MATLAB 是以 i 或 j 字元来代表虚部复数运算的。

一个复数可表示为：$x = a + bi$，其中 a 称为实部，b 称为虚部。

也可写成复指数形式：$x = re^{i\theta}$。其中 r 称为复数的模，又记为 $|x|$；θ 称为复数的幅角，又记为 $Arg(x)$，且满足如下关系：

$$r = \sqrt{a^2 + b^2}\ ,\quad \tan\theta = \frac{b}{a}$$

具体采用哪种表示方法取决于实际问题。一般而言，第一种问题适合处理复数的代数运算，第二种方法适合处理复数旋转等涉及幅角改变的问题。

一个复数也可以看做是关于实部和虚部的符号函数。因此，既可以直接构造复数，也可以用符号函数构造复数。下面分别介绍这两种构造方法。

（1）用直接法构造两种形式的复数

直接法就是利用符号 i 或 j 来表示复数单位，将复数看做完整的一个表达式输入。其

具体形式也有两种，可以用实部虚部形式表示，也可以用复指数形式表示。

（2）用符号函数法构造两种形式的复数

所谓符号函数法就是将复数看成是函数形式，其实部和虚部看做自变量，用 syms 来构造，用 subs 对符号函数中自变量进行赋值。

例 2-7 复数构造实例。试分别使用上述两种方法，在 MATLAB 中构造复数 $x=-1+i$。

解： 在 MATLAB 命令窗口中输入：

```
x1=-1+i  %直接法构造,实部虚部形式
x2=sqrt(2)*exp(i*(3*pi/4)) %直接法构造,复指数形式
%符号函数法构造,实部虚部形式
syms a b real;      %声明 a, b 为实数型
x3=a+b*i %实部虚部形式复数的符号表达
subs(x3,{a,b},{-1,1}) %代入具体值
%符号函数法构造,复指数形式
syms r ct real; %声明 r, ct 为实数型
x4=r*exp(ct*i); %复指数形式复数的符号表达
subs(x4,{r,ct},{sqrt(2),3*pi/4})%代入具体值
```

输出结果为：

```
x1 =  -1.0000 + 1.0000i
x2 =  -1.0000 + 1.0000i
x3 =  -1.0000 + 1.0000i
x4 =  -1.0000 + 1.0000i
```

MATLAB 中矩阵运算贯穿始终，复数矩阵也是其中的一个重要方面。由于复数矩阵的每个元素都是复数，所以对应的复数创建矩阵也有两种，即直接创建和利用符号函数创建。

复数矩阵的直接创建有两种方法，一种是由复数元素构造复数矩阵，另一种是利用两个矩阵分别做实部和虚部构造新的复数矩阵。下面举例说明复数矩阵的直接创建，另一种方法类比很容易实现，在此不作赘述。

（1）由复数元素构造复数矩阵

复数矩阵的每个元素都是复数，将复数作为矩阵元素并按照矩阵格式进行填充就得到了复数矩阵。

（2）由实矩阵创造复数矩阵

由两个分别作实部和虚部的同维矩阵构造复数矩阵。

例 2-8 复数矩阵构造实例。构造复数矩阵 $\begin{bmatrix} 1+i & 1+2i & 1+3i \\ 1-i & 1-2i & 1-3i \end{bmatrix}$。

解： 可直接输入各元素来构造矩阵，此处通过其复指数形式构造。

在 MATLAB 命令窗口中输入：

```
%直接输入各元素来构造
A1=[sqrt(2)*exp((pi/4)*i) 1+2i 1+3i;sqrt(2)*exp((-pi/4)*i) 1-2i 1-3i] %
%由实矩阵构
A2re=[1 1 1;1 1 1]; A2im=[1 2 3;-1 -2 -3]; A2=A2re+A2im*i
```

输出结果 A1 和 A2 均为:

```
   1.0000 + 1.0000i   1.0000 + 2.0000i   1.0000 + 3.0000i
   1.0000 - 1.0000i   1.0000 - 2.0000i   1.0000 - 3.0000i
```

2.4.2 复数的绘图

对于复数函数的绘图主要有两种形式。一种是直角坐标图，即分别以复数的实部和虚部为坐标作出复数的表示图；另一种为极坐标图，即分别以复数的模和幅角为坐标作图。

MATLAB 提供了绘制极坐标图的函数 polar，它还可绘制出极坐标栅格线，其常用的调用格式为：

```
polar(theta,rho)
```

其中，theta 为极坐标极角，rho 为极坐标矢径。

例 2-9 复数函数绘图实例。作出函数 $y = t + it\sin(t)$ 在两种坐标下的表示图。

解： 直角坐标图——以实部为横坐标，以虚部为纵坐标；极坐标图——以模为极半径，以幅角为极角，在 MATLAB 命令窗口中输入以下命令，最终显示结果如图 2-1 所示。

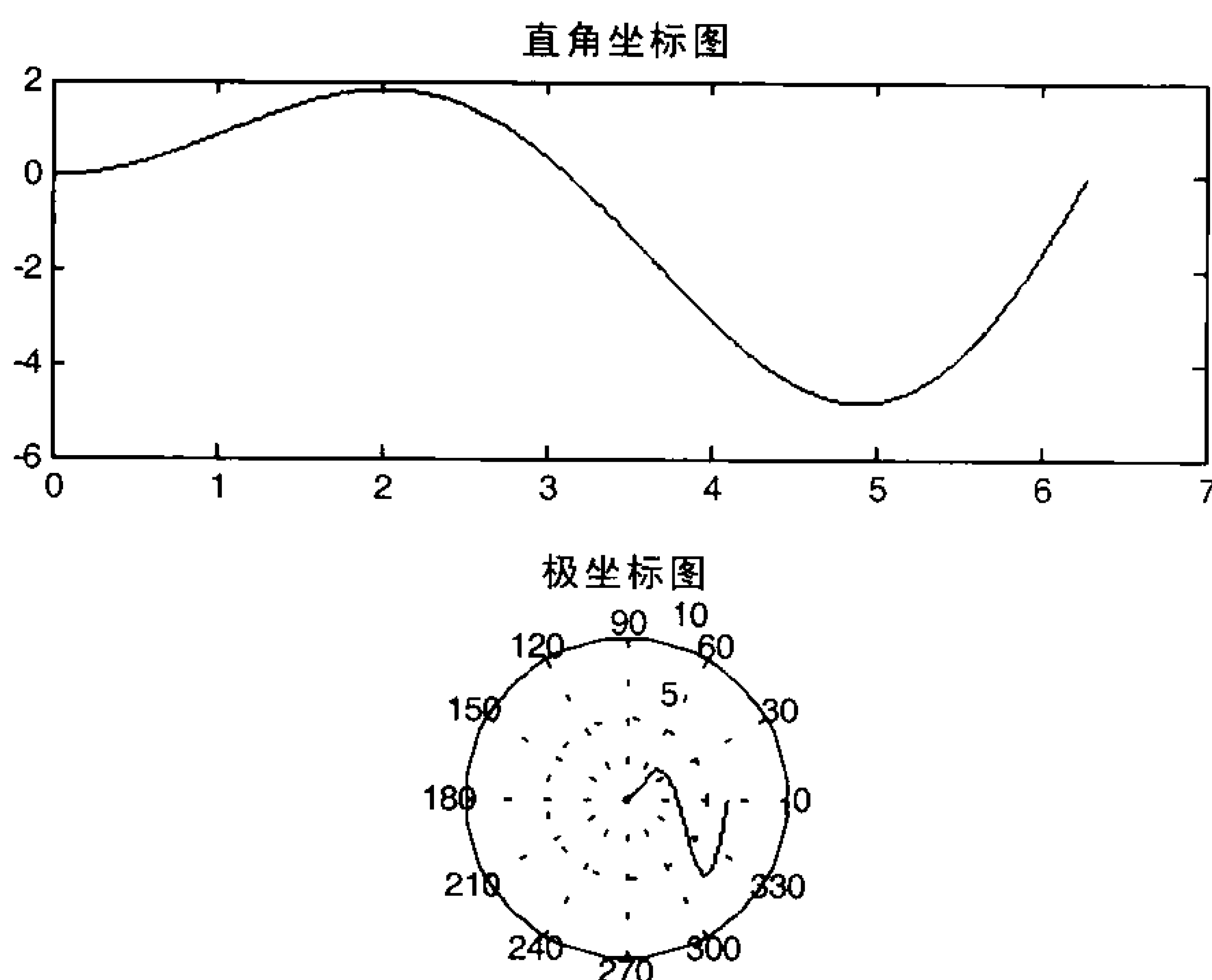


图 2-1 极坐标下复数的表示

```
t=0:0.01:2*pi; y=t+i*t.*sin(t); %直角坐标表示
r=abs(y);bdelta=angle(y); %极坐标表示
subplot(2,1,1)
plot(y)    %绘制直角坐标图
title('直角坐标图') ; subplot(2,1,2)
polar(delta,r)  %绘制极坐标图
title('极坐标图')
```

2.4.3 复数的操作函数

对于一个形如 $x=a+bi$ 的复数，我们通常希望能够了解它自身的结构性质，包括实部、虚部、模和幅角等。MATLAB 提供了方便的操作函数，如表 2.11 所示。下面利用实例来说明各个函数的调用方法。

表 2.11 常用矩阵分解运算函数

函 数 名	功 能	函 数 名	功 能
real(A)	求复数或复数矩阵 A 的实部	abs(A)	求复数或复数矩阵 A 的模
imag(A)	求复数或复数矩阵 A 的虚部	angle(A)	求复数或复数矩阵 A 的相角，单位为弧度
conj(A)	求复数或复数矩阵 A 的共轭		

在 MATLAB 中，复数的基本运算和实数相同，都是使用相同的函数。比如复数或复数矩阵 x 除以 y，运算命令也是 x/y，与实数运算一样。因此，复数基本运算的内容此处不再赘述。

2.4.4 留数的基本运算

留数的定义：设 a 是 $f(z)$ 的孤立奇点，C 是 a 的充分小的邻域内一条把 a 点包含在其内部的闭路，积分 $\frac{1}{2\pi}\int_C f(z)\mathrm{d}z$ 称为 $f(z)$ 在 a 点的留数或残数，记作 $\mathrm{Res}[f(z),a]$。

留数定理：如果函数 $f(z)$ 在闭路 C 上解析，在 C 的内部除去 n 个孤立奇点 $a_1,a_2,\cdots a_n$ 外也解析，则闭路上的积分满足：

$$\int_C f(z)\mathrm{d}z = 2\pi i\sum_{k=1}^{n}\mathrm{Res}[f(z),a_k]$$

如果能够求得函数 $f(z)$ 在各极点的留数的显式表达，则该闭路积分很容易获得。

由罗朗展开：若 a 是 $f(z)$ 的 m 重极点，则函数在该点的留数可以表示为：

$$\mathrm{Res}[f(z),a]=\frac{1}{(m-1)!}\lim_{z\to a}\frac{\mathrm{d}^{m-1}}{\mathrm{d}z^{m-1}}[(z-a)^m f(z)]。$$

至此，通过留数定理可以把闭路积分转化为简单的代数计算。

由于在工程中遇到的 $f(z)$ 多数情况下为有理分式，所以可表示为如下形式：

$$\frac{a_n z^n + a_{n-1}z^{n-1}+\ldots+a_1 z+a_0}{b_m z^m + b_{m-1}z^{m-1}+\ldots+b_m z+b_0}$$

函数 residue 可以求得该有理式的留数，residue 的返回参数有 3 个，分别对应留数向量、极点向量和高阶项。

调用格式：[r, p, k] = residue($[a_n \quad a_{n-1} \quad \ldots \quad a_0]$, $[b_n \quad b_{n-1} \quad \ldots \quad b_0]$)

如果仅对留数感兴趣，后两个参数可省略。

2.5 符号运算

MATLAB 提供了符号数学工具箱（Symbolic Math Toolbox），大大增强了 MATLAB 的功能。符号数学工具箱的特点为：

（1）符号数学工具箱适用于广泛的用途，而不是针对一些特殊专业或专业分支；

（2）符号数学工具箱使用字符串来进行符号分析，而不是基于数组的数值分析。

2.5.1 符号运算概述

符号数学工具箱是操作和解决符号表达式的符号数学工具箱（函数）集合，有复合、简化、微分、积分以及求解代数方程和微分方程的工具。另外还有一些用于线性代数的工具，求解逆、行列式、正则形式的精确结果，找出符号矩阵的特征值而没有由数值计算引入的误差。工具箱还支持可变精度运算，即支持符号计算并能以指定的精度返回结果。

符号运算与数值运算的主要区别如下：

（1）数值运算中必须先对变量赋值，然后才能参与运算；

（2）符号运算无须事先对独立变量赋值，运算结果以标准的符号形式表达。

符号运算的运算对象可以是没赋值的符号变量，可以获得任意精度的解。

1. 符号表达式

符号表达式是代表数字、函数、算子和变量的 MATLAB 字符串，或字符串数组。不要求变量有预先确定的值，符号方程式是含有等号的符号表达式。符号算术是使用已知的规则和给定符号恒等式求解这些符号方程的实践，它与代数和微积分所学到的求解方法完全一样。符号矩阵是数组，其元素是符号表达式。

MATLAB 在内部把符号表达式表示成字符串，与数字变量或运算相区别；否则，这些符号表达式几乎完全像基本的 MATLAB 命令。

2. 创建符号对象

MATLAB 提供了两个建立符号对象的函数：sym 和 syms，两个函数的用法介绍如下。

（1）sym 函数

sym 函数用来建立单个符号量，一般调用格式为：

```
符号量名=sym('符号字符串')
```

该函数可以建立一个符号量，符号字符串可以是常量、变量、函数或表达式。应用 sym 函数还可以定义符号常量。

（2）syms 函数

函数 sym 一次只能定义一个符号变量，使用不方便。MATLAB 提供了另一个函数 syms，一次可以定义多个符号变量。

syms 函数的一般调用格式为：

```
syms 符号变量名1 符号变量名2 … 符号变量名n
```

用这种格式定义符号变量时不要在变量名上加字符串分界符(')，变量间用空格而不要用逗号分隔。

含有符号对象的表达式称为符号表达式，建立符号表达式有以下 3 种方法：

- 利用单引号来生成符号表达式；
- 用 sym 函数建立符号表达式；
- 使用已经定义的符号变量组成符号表达式。

例 2-10 符号表达式创建实例。

解：在 MATLAB 窗口，输入下列命令：

```
U=sym('3*x^2+5*y+2*x*y+6')        %定义符号表达式
syms x y;                         %建立符号变量 x、y
V=3*x^2+5*y+2*x*y+6               %定义符号表达式 V
2*U-V+6                           %求符号表达式的值
```

3. 符号矩阵

符号矩阵也是一种符号表达式，所以前面介绍的符号表达式运算都可以在矩阵意义下进行。但应注意这些函数作用于符号矩阵时，是分别作用于矩阵的每一个元素。

通过 sym 函数创立符号矩阵，矩阵元素可以是任何不带等号的符号表达式，各符号表达式的长度可以不同，矩阵元素之间可用空格或逗号分隔。

例如，在命令窗口中输入 A = sym('[a , 2*b ; 3*a , 0]') ，就完成了一个符号矩阵 $A=\begin{bmatrix} a & 2b \\ 3a & 0 \end{bmatrix}$ 的创建，输出的结果为：

```
A = [ a, 2*b]
    [3*a, 0]
```

需要注意的是：符号矩阵的每一行的两端都有方括号，这是与 MATLAB 数值矩阵的一个重要区别。

在 MATLAB 中，数值矩阵不能直接参与符号运算，必须先转化为符号矩阵。

（1）将数值矩阵转化为符号矩阵

函数调用格式为：sym(数值矩阵)

（2）将符号矩阵转化为数值矩阵

函数调用格式：numeric(A)

由于符号矩阵是一个矩阵，所以符号矩阵还能进行有关矩阵的运算。MATLAB 还有一些专用于符号矩阵的函数，这些函数作用于单个的数据无意义。例如

transpose(s)：返回 s 矩阵的转置矩阵。

determ(s)：返回 s 矩阵的行列式值。

其实，许多应用于数值矩阵的函数，如 diag、triu、tril、inv、det、rank、eig 等，也可直接应用于符号矩阵。

4. 符号表达式的四则运算

符号表达式的四则运算比较简单，常用的函数介绍如下。

- factor(S)：对 S 分解因式，S 是符号表达式或符号矩阵。
- expand(S)：对 S 进行展开，S 是符号表达式或符号矩阵。
- collect(S)：对 S 合并同类项，S 是符号表达式或符号矩阵。
- collect(S,v)：对 S 按变量 v 合并同类项，S 是符号表达式或符号矩阵。
- simplify(S)：应用函数规则对 S 进行化简。
- simple(S)：调用 MATLAB 的其他函数对表达式进行综合化简，并显示化简过程。

2.5.2 常用的符号运算

符号变量和数字变量之间可转换，也可以用数字代替符号得到数值。符号运算种类非常多，常用的符号运算有代数运算、积分和微分运算、极限运算、级数求和、进行方程求解等。

出于篇幅的考虑，下面仅对控制系统中常用的符号运算进行介绍，至于其他的符号运算，读者可通过 MATLAB 的帮助文档或其他关于符号函数工具箱的书籍进行学习，此处不再赘述。

常用的符号运算有微积分、拉普拉斯变换和 Z 变换等积分变换，下面分别进行介绍。

diff 是求微分最常用的函数，其输入参数既可以是函数表达式，也可以是符号矩阵。

常用的格式是：diff(f, x, n)，表示 f 关于 x 求 n 阶导数。

int 是求积分最常用的函数，其输入参数可以是函数表达式。

常用的格式是：int(f, r, x0, x1)。其中，f 为所要积分的表达式，r 为积分变量，若为定积分，则 x0 与 x1 为积分上下限。

例 2-11 符号运算实例 1。已知表达式 f=sin（ax），分别对其中的 x 和 a 求导。

解： 输入如下 MATLAB 程序代码。

```
>> syms a x                  %定义符号变量 a 和 x
>> f=sin(a*x)                %创建函数 f
>> dfx=diff(f, x)            %对 x 求导
>> dfa=diff(f, a)            %对 a 求导
```

运行程序，输出结果如下所示：

```
f =sin(a*x)
dfx =cos(a*x)*a              %f 对 x 求导的结果
dfa =cos(a*x)*x              %f 对 a 求导的结果
```

例 2-12 符号运算实例 2。已知表达式 $f = x\log(1+x)$，求对 x 的积分和 x 在(0,1)上的积分值。

解： 输入如下 MATLAB 程序代码。

```
>>syms x                    %定义符号变量 x
>>f=x*log(1+x)              %创建函数 f
>>int1=int(f,x)             %对 x 积分
>>int2=int(f,x,0,1)         %求[0,1]区间上的积分
```

运行程序，输出结果如下所示：

```
int1 =1/2*(1+x)^2*log(1+x)+3/4+1/2*x-1/4*x^2-(1+x)*log(1+x)   %积分表达式
int2 =1/4                   %积分值
```

2.6 MATLAB 中的数据精度

2.6.1 MATLAB 的数据类型

MATLAB 的基本数值数据类型有两类：整数型和浮点型。

整数型数据按照表示范围可分为 int8、int16、int32、int64、uint8、uint16、uint32、uint64 八种类别，其中每种类型表示的数据范围如表 2.12 所示：

表 2.12 MATLAB 整数的表示范围

数据类型	表示范围	数据类型	表示范围
int8	$-2^7 \sim 2^7-1$	uint8	$0 \sim 2^8-1$
int16	$-2^{15} \sim 2^{15}-1$	uint16	$0 \sim 2^{16}-1$
int32	$-2^{31} \sim 2^{31}-1$	uint32	$0 \sim 2^{32}-1$
int64	$-2^{63} \sim 2^{63}-1$	uint64	$0 \sim 2^{64}-1$

当数据超过类型的表示范围时，MATLAB 将数据表示成该类型的最大值或最小值，如下面的例题所示。

例 2-13 数据类型使用实例。将 234 与-234 表示成 int8 类型。

解： 在 MATLAB 命令窗口输入下列命令。

```
>> int8(234)
ans =  127
>> int8(-234)
ans = -128
```

由于 int8 表示的整数范围为 $-2^7 \sim 2^7-1$，而 $234>127$，所以输出结果为 int8 能够表示的最大整数 127，同理由于 $-234>-128$，所以输出结果为 int8 能够表示的最大整数-128。对于其他的整数类型都是一样的。

浮点型数据按照表示范围可分为单精度和双精度两种类型，其中每种类型表示的数据

范围如表 2.13 所示。

表 2.13 MATLAB 浮点数据的表示范围

浮点类型	表示范围	浮点类型	表示范围
单精度	$-2^{128} \sim -2^{-126}$, $2^{-126} \sim 2^{128}$	双精度	$-2^{1024} \sim -2^{-1022}$, $2^{-1022} \sim 2^{1024}$

表 2.13 的数据来自于 MATLAB 的帮助文件，这些数据都是遵循 IEEE 标准得出的数据。从表 2.13 可以看出，双精度数据能够表示的最小实数为 2^{-1022}，即用双精度表示数据的精度为 2^{-1022}。当然 MATLAB 内部有专门的函数获得这些数据。

例 2-14 数据类型精度范围使用实例。获取双精度数据的表示范围。

解：在 MATLAB 命令窗口输入下列命令。

```
>> str = '双精度数据的表示范围为:\n\t%g to %g \n\t %g to  %g';
>> sprintf(str, -realmax, -realmin, realmin, realmax)
ans =
```

双精度数据的表示范围为:

```
-1.79769e+308 to -2.22507e-308
 2.22507e-308 to  1.79769e+308
```

MATLAB 中默认的数据类型为双精度类型。

2.6.2 MATLAB 的数值精度

MATLAB 的数值精度也就是 MATLAB 能够表示的最小实数，任何一个绝对值小于 MATLAB 的数值精度的实数都会被当成 0 处理。经过实验，可以知道，在 MATLAB 7.0 版本中，MATLAB 能够表示的最小实数为 2^{-1074}，任何绝对值小于 2^{-1074} 的实数，MATLAB 都将其视为 0。

例 2-15 MATLAB 数值精度实例。MATLAB 7.0 的数值精度。

解：在 MATLAB 命令窗口输入下列命令。

```
>> >> x = 2^(-1074)
x =  4.9407e-324
>> x = 2^(-1075)
x =      0
>> x == 0
ans =   1
```

从上面的例子可以看出，由于 $2^{-1075} < 2^{-1074}$，MATLAB 7.0 视其为 0，而且通过和 0 的比较可以看出，结果确实为真。

因此，如果在 MATLAB 程序中要判断某个实数是否等于 0，最可靠的办法是将其与 2^{-1074} 相比较，看它的绝对值是否小于等于 2^{-1074}。

当然对于具体的问题可以根据问题的需要和精度，采取不同的比较对象，例如

MATLAB 中有一个内置常量 eps，其值为 2.2204×10^{-16}，它可以作为实数是否等于 0 的一个比较对象。

2.6.3 MATLAB 的显示精度

MATLAB 的显示精度是指 MATLAB 显示的有效位数。MATLAB 中的显示精度是可以修改的，显示精度修改了，原来的数据并没有变，只是数据在 MATLAB 命令窗口中显示的有效位数不同而已。

MATLAB 中有三个函数可以设置显示精度：format、vpa 和 digits，它们使用简单，具体用法可参考 MATLAB 帮助文档。

例 2-16 MATLAB 显示精度实例。MATLAB 的显示精度。

解：在 MATLAB 命令窗口输入下列命令。

```
>> x = 1/3                  %MATLAB 默认的显示精度
x =    0.3333
>> format long;             %设置为 long 显示精度
>> x = 1/3
x =   0.33333333333333
>> format rational;         %显示为小数形式
>> x = 1/3
x =     1/3
>> digits(10);              %显示 10 位有效数字
>> vpa(1/3)
ans = .3333333333
>> vpa(100/3,20)            %显示 20 位有效数字
ans = 33.333333333333333333
```

2.7 MATLAB 常用绘图命令

MATLAB 提供了强大的图形用户界面，在许多应用中，常常要用绘图功能来实现数据的显示和分析，包括二维图形和三维图形。在控制系统仿真中，也常常用到绘图，如绘制系统的响应曲线、根轨迹或频率响应曲线等。

MATLAB 提供了丰富的绘图功能，在命令窗口中输入“help graph2d”可得到所有画二维图形的命令；输入“help graph3d”可得到所有画三维图形的命令。

下面主要介绍常用的二维图形命令的使用方法，三维图形命令的使用方法与此类似。至于这些命令的全部用法请参考在线帮助系统。

1．基本的绘图命令

plot(x1, y1, option1, x2, y2, option2, …)：x1 与 y1 给出的数据分别为 x 轴与 y 轴坐标值，option1 为选项参数，以逐点连折线的方式绘制 1 个二维图形；同时类似地绘制第二个二维图形。

这是plot命令的完全格式，在实际应用中可以根据需要进行简化。比如plot(x, y)、plot(x, y, option)，选项参数 option 定义了图形曲线的颜色（用颜色英文单词的第一个字母表示，例如 r 表示红色、g 表示绿色、b 表示蓝色）、线型（例如#、*等）及标示符号，它由一对单引号括起来。

2. 图形窗口处理命令

常用的选择图形窗口的命令有：

- 打开不同的图形窗口命令 figure

figure(1)；figure(2)；…；figure(n)，它用来打开不同的图形窗口，以便绘制不同的图形。

- 图形窗口拆分命令 subplot

subplot(m, n, p)：分割图形显示窗口，m 表示上下分割个数，n 表示左右分割个数，p 表示子图编号。

3. 坐标轴相关的命令

在默认情况下 MATLAB 自动选择图形的横、纵坐标的比例，当然也可以用 axis 命令控制，常用的命令介绍如下。

- axis([xmin xmax ymin ymax])：[xmin xmax ymin ymax]中分别给出 x 轴和 y 轴的最大值、最小值。
- axis equal：x 轴和 y 轴的单位长度相同。
- axis square：图框呈方形。
- axis off：清除坐标刻度。

在某些应用中，还会用到半对数坐标轴，MATLAB 中常用的对数坐标绘制命令介绍如下。

- semilogx：绘制以 x 轴为对数坐标（以 10 为底）、y 轴为线性坐标的半对数坐标图形。
- semilogy：绘制以 y 轴为对数坐标（以 10 为底）、x 轴为线性坐标的半对数坐标图形。
- loglog：绘制全对数坐标绘图，即 x、y 轴均为对数坐标（以 10 为底）。

4. 文字标示命令

常用的文字标示命令介绍如下。

- text(x, y, '字符串')：在图形的指定坐标位置(x, y)处标示单引号括起来的字符串。
- gtext('说明文字')：利用鼠标在图形的某一位置标示说明文字。执行完绘图命令后再执行 gtext('说明文字') 命令，就可在屏幕上得到一个光标，然后用鼠标选择说明文字的位置。
- title('字符串')：在所画图形的最上端显示说明该图形标题的字符串；
- xlabel('字符串')、ylabel('字符串')、zlabel('字符串')：设置 x、y、z 坐标轴的名称。

输入特殊的文字需要用反斜杠（\）开头。

- legend('字符串 1', '字符串 2', …, '字符串 n')：在屏幕上开启一个小视窗，然后依据绘图命令的先后次序，用对应的字符串区分图形上的线。

5. 在图形上添加或删除栅格命令

常用的栅格操作命令介绍如下。

- grid：给图形加上栅格线。
- grid on：给当前坐标系加上栅格线。
- grid off：从当前坐标系中删去栅格线。
- grid：交替转换命令，即执行一次，转变一个状态（相当于 grid on、grid off）。

6. 图形保持或覆盖命令

常用的图形保持和覆盖的命令介绍如下。

- hold on：把当前图形保持在屏幕上不变，同时允许在这个坐标内绘制另外一个图形。
- hold off：使新图覆盖旧图。

hold 命令可以保持当前的图形，并且防止删除和修改比例尺。

hold 命令是一个交替转换命令，即执行一次，转变一个状态（相当于 hold on、hold off）。



MATLAB 默认为 hold off，这时的操作会修改图形的属性的，因此需要在 plot 之前加上 hold on。

7. 应用型绘图命令

应用型绘图命令常用于数值统计分析或离散数据处理，常用的应用型绘图命令介绍如下。

- bax(x, y)：绘制对应于输入 x 和输出 y 的高度条形图。
- hist(y, x)：绘制 x 在以 y 为中心的区间中分布的个数条形图。
- stairs(x, y)：绘制 y 对应于 x 的梯形图。
- stem(x, y)：绘制 y 对应于 x 的散点图。



对于图形的属性编辑同样可以在图形窗口上直接进行，但图形窗口关闭之后编辑结果不会保存。

例 2-17 绘图命令使用实例。绘制[0,4π]区间上的 x1=10sint 和 x2=5cost 曲线，并要求：

（1）x1 曲线的线形为点划线、颜色为红色、数据点标记为加号；x2 曲线的虚线、颜色为蓝色、数据点标记为星号；

（2）标示坐标轴的显示范围和刻度线、添加栅格线；

（3）标注坐标轴名称、标题、相应文本。

解： MATLAB 程序代码如下所示。

```
close all  %关闭打开了的所有图形窗口
clc  %清屏命令
clear    %清除工作空间中所有变量
t=[0:pi/20:4*pi];     %定义时间范围
hold on   %允许在同一坐标系下绘制不同的图形
axis([0 4*pi -10 10]) %横轴范围[0,4π], 纵轴范围[-10,10]
plot(t, 10*sin(t), 'r+:') %线形为点划线、颜色为红色、数据点标记为加号
plot(t, 5*cos(t),'b*--') %线形为虚线、颜色为蓝色、数据点标记为星号
xlabel('时间t'); ylabel('幅值x')    %标注横、纵坐标轴
title('简单绘图实例') %添加图标题
legend('x1=10sint:点划线','x2=5cost:虚线')  %添加文字标注
gtext('x1'); gtext('x2') %利用鼠标在图形标示曲线说明文字
grid on    %在所画出的图形坐标中添加栅格, 注意用在 plot 之后
```

运行后，输出结果如图 2-2 所示。

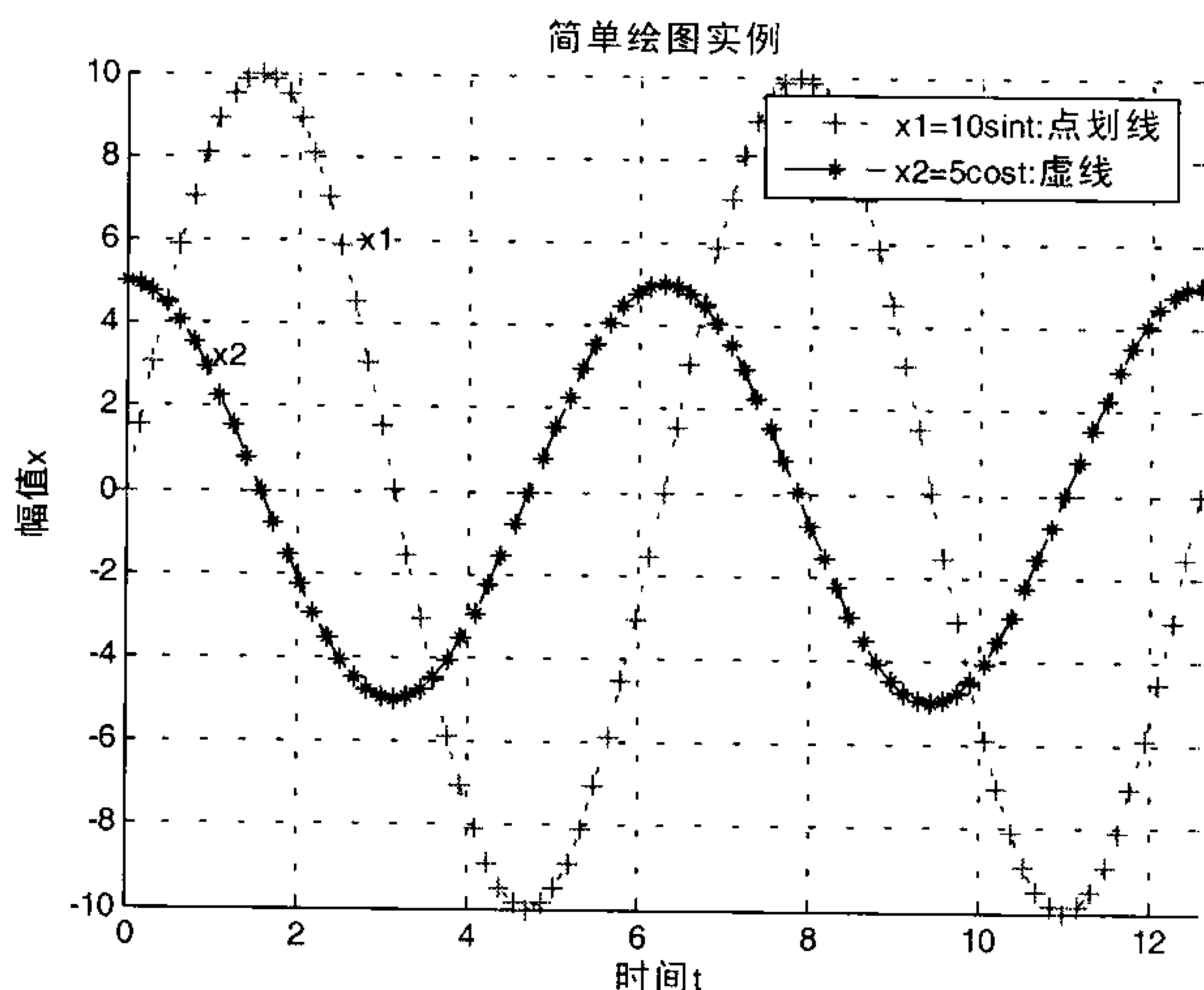


图 2-2 例 2-17 的输出图

2.8 小结

MATLAB 语言具有强大的数值计算和数据可视化功能，熟练掌握和运用这些功能是发挥 MATLAB 强大功能的基础。

第 3 章　MATLAB 程序设计基础

在 MATLAB 中，除了可以在命令窗口中输入命令逐句执行外，也可以和其他形式的 C、FORTRAN 等高级语言一样采用编程的方式，称为 M 文件编程。

读者首先应掌握 MATLAB 程序设计的基本方法，不断实践，逐步将其强大的功能应用到科学计算及其他领域的学习和应用中去。

3.1 MATLAB 编程概述

MATLAB 不仅是一种功能强大的高级语言，而且是一个集成的交互式开发环境，用户可以通过 MATLAB 提供的编辑调试器编写和调试 MATLAB 代码。

MATLAB 提供了代码书写和调试的集成开发环境，用户可以在 MATLAB 的代码编辑调试器中完成书写和调试过程。单击 MATLAB 主界面的"新建"工具按钮或者单击文件菜单（File）"新建子菜单"（New）的"M-File"项，就可以打开 MATLAB 代码编辑-调试器，其空白界面如图 3-1 所示。

用户也可以在命令窗口通过 edit filename 命令打开已存在的 M 文件进行编辑调试。

从图 3-1 可见，MATLAB 能够根据 M 文件内容区别是脚本 M 文件还是函数 M 文件，并且在整个编辑过程中追踪光标位置（如图 3-1 底部的"Ln 1 Col 1"表示当前光标处在第一行的第一列），这对于准确快速定位当前编辑和修改位置是很方便有用的。

开发 MATLAB 程序一般需要经历代码编写、调试、优化几个阶段。

在编写代码时，要及时保存阶段性成果，可以通过 File 菜单的 Save 项或者保存工具按钮保存当前 M 文件。

完成代码书写之后，要试运行代码看看有没有运行错误，然后根据针对性的错误提示对程序进行修改。

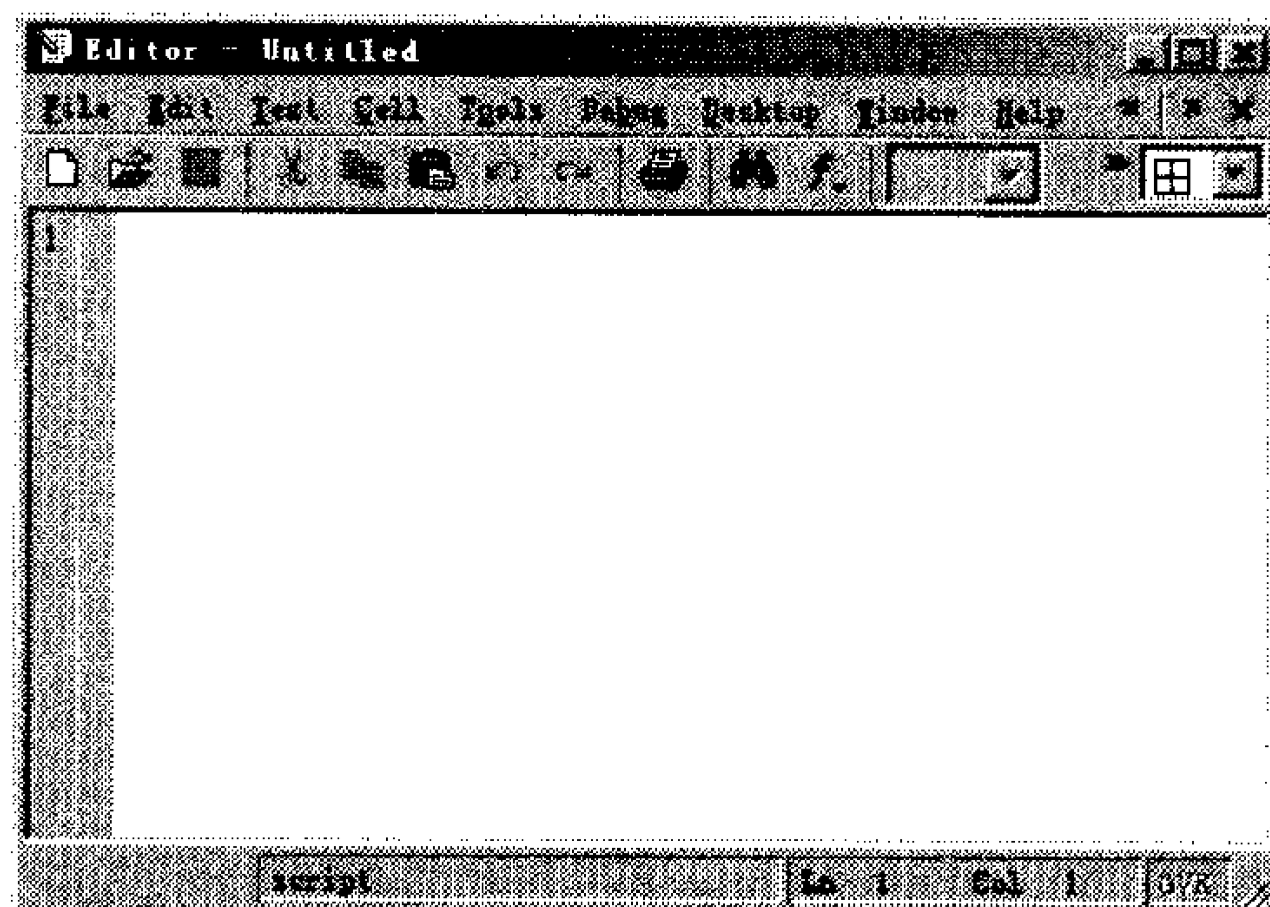


图 3-1　MATLAB 代码编辑-调试器

运行脚本 M 文件，只需要在命令窗口中输入其文件名，然后按回车键，或通过 Debug 菜单的 Run 项，或按快捷键“F5”完成。

运行函数 M 文件，需要通过命令窗口传递输入参数来调用。除了一些很简单的代码，大部分情况下用户都可能遇到程序报错的问题，这就需要对代码进行调试纠错，一般需要通过 Debug 菜单下的子项辅助完成，包括设置断点、逐步运行等项。

当程序运行无误后，还要考虑程序性能是否可以改进。

MATLAB 提供了 M-Lint 和 Profiler 工具，能够辅助用户分析代码运行中时间消耗的细节和可能需要改变的编程细节，如循环赋值前没有预定义数组，用循环去实现可以用数组函数实现的运算等。这些工具都在 Tools 菜单下设置了子菜单。

3.2 MATLAB 程序设计原则

MATLAB 程序的基本设计原则如下所述。

- 百分号“%”后面的内容是程序的注解，要善于运用注解使程序更具可读性。
- 养成在主程序开头用 clear 指令清除变量的习惯，以消除工作空间中其他变量对程序运行的影响，但注意在子程序中不要用 clear。
- 参数值要集中放在程序的开始部分，以便维护。要充分利用 MATLAB 工具箱提供的指令来执行所要进行的运算，在语句行之后输入分号使其及中间结果不在屏幕上显示，以提高执行速度。
- input 指令可以用来输入一些临时的数据；而对于大量参数，则通过建立一个存储参数的子程序，在主程序中通过子程序的名称来调用。
- 程序尽量模块化，即采用主程序调用子程序的方法，将所有子程序合并在一起来执行全部的操作。
- 充分利用 Debugger 来进行程序的调试（设置断点、单步执行、连续执行），并利用其他工具箱或图形用户界面（GUI）的设计技巧，将设计结果集成到一起。
- 设置好 MATLAB 的工作路径，以便程序运行。
- MATLAB 程序的基本组成结构如下所示。

MATLAB 程序的基本组成结构
%说明
清除命令：清除 workspace 中的变量和图形（clear,close）
定义变量：包括全局变量的声明及参数值的设定
逐行执行命令：指 MATLAB 提供的运算指令或工具箱提供的专用命令
…
…
…
控制循环：包含 for，if then，switch，while 等语句
逐行执行命令

```
…
…
end
绘图命令：将运算结果绘制出来
```

当然，更复杂的程序还需要调用子程序，或与其他应用程序相结合。

3.3 M 文件

M 文件是包含 MATLAB 代码的文件。

1. M 文件的类型

M 文件按其内容和功能可以分为脚本 M 文件和函数 M 文件两大类。

（1）脚本 M 文件

它是许多 MATLAB 代码按顺序组成的命令序列集合，不接受参数的输入和输出，与 MATLAB 工作空间共享变量空间。

它一般用来实现一个相对独立的功能，比如对某个数据集进行某种分析、绘图，求解某个已知条件下的微分方程等。用户可以通过在命令窗口中直接输入文件名来运行脚本 M 文件。

通过脚本 M 文件，用户可以把为实现一个具体功能的一系列 MATLAB 代码书写在一个 M 文件中，每次只需要输入文件名即可运行脚本 M 文件中的所有代码。

（2）函数 M 文件

它也是为了实现一个单独功能的代码块，但与脚本 M 文件不同的是函数 M 文件需要接受参数输入和输出，函数 M 文件中的代码一般只处理输入参数传递的数据，并把处理结果作为函数输出参数返回给 MATLAB 工作空间中的指定接收变量。

因此，函数 M 文件具有独立的内部变量空间。在执行函数 M 文件时，要指定输入参数的实际取值，而且一般要指定接收输出结果的工作空间变量。

MATLAB 提供的许多函数就是用函数 M 文件编写的，尤其是各种工具箱中的函数，用户可以打开这些 M 文件来查看。实际上，对于特殊应用领域的用户，如果积累了充分的专业领域应用的函数，就可以组建自己的专业领域工具箱了。

通过函数 M 文件，用户可以把实现一个抽象功能的 MATLAB 代码封装成一个函数接口，在以后的应用中重复调用。

2. M 文件的结构

图 3-2 显示的是 MATLAB 提供的一个函数 M 文件的全部内容，图中清楚地显示了一般的 M 文件包括的各部分结构。

从图 3-2 可以看到，MATLAB 中 M 文件一般包括以下五部分结构。

- 函数声明行（Function Definition Line），这一行只出现在函数 M 文件的第一行，通

过 function 关键字表明此文件是一个函数 M 文件，并指定函数名、输入和输出参数，如图 3-2 中的第一行。

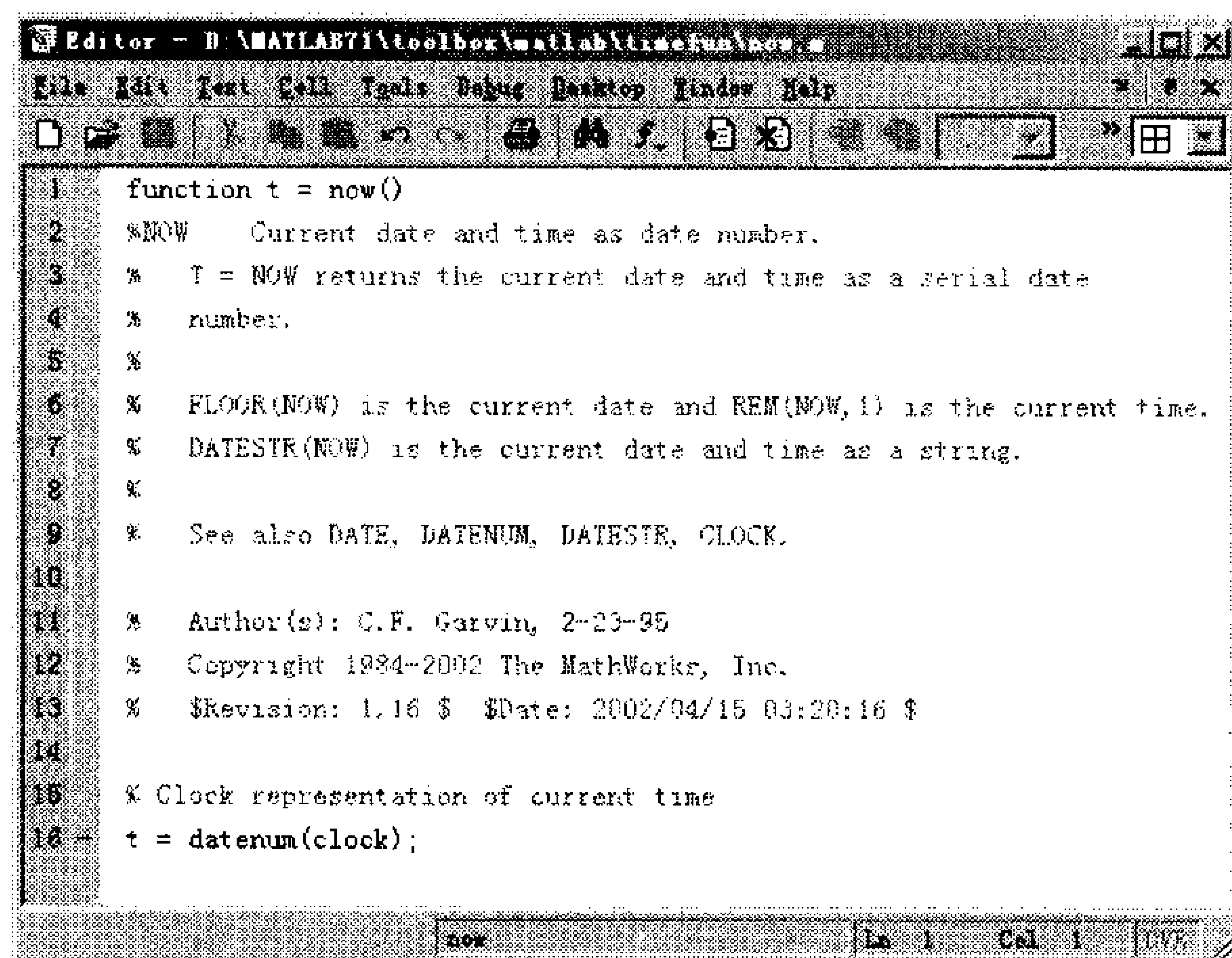


图 3-2 M 文件的一般结构

- H1 行，这是帮助文字的第一行（the first help text line），给出 M 文件帮助最关键的信息。当用 lookfor 查找某个单词相关的函数时，lookfor 只在 H1 行中搜索是否出现指定单词，如图 3-2 中的第 2 行。
- 帮助文字，这部分对 M 文件有更加详细的说明，经常解释 M 文件实现的功能，M 文件中出现的各变量、参数的意义，以及创作版权信息等。如图 3-2 中的第 13 行。当获取一个 M 文件的帮助时，H1 行和帮助文字部分同时显示。
- M 文件正文，这是 M 文件实现功能的 MATLAB 代码部分，通常包括运算、赋值等指令。图 3-2 的例子中只有第 16 行，但一般都由多行组成。
- 注释部分，这部分出现的位置比较灵活，主要是用来注释 M 文件正文的具体运行过程，以方便阅读和修改，经常穿插在 M 文件正文中间。

图 3-2 的例子中的第 15 行就是注释说明正文第 16 行的意义。注释一般都是针对接下来的一段正文代码的，常见的 M 文件中也经常包括多行注释。

3. M 文件的创建

虽然一般脚本 M 文件可以包括上述五部分结构中除去“函数声明行”以外的四部分，但在实际应用中，脚本 M 文件经常仅仅由 M 文件正文和注释部分构成。正文部分实现功能，注释部分则给出每一块代码的功能说明。下面通过实例讲述脚本 M 文件的创建。

例 3-1 M 文件创建实例。建立一个命令文件，将变量 *a*、*b* 的值互换。

解： 首先打开 M 文件编辑器，输入以下程序：

```
a=1:9;  b=[11,12,13;14,15,16;17,18,19];
c=a;  a=b;  b=c;
a
b
```

然后保存文件名为“31.m”即完成了文件的建立。

在 MATLAB 的命令窗口中输入 31，将会执行该命令文件。

```
>> 31
a = 11    12    13
    14    15    16
    17    18    19
b =  1     2     3     4     5     6     7     8     9
```

函数 M 文件的命名一般习惯和函数名一致，比如图 3-2 中函数声明行 function t=now()，表明函数名为 now，因此此函数 M 文件一般保存为 now.m，可以通过 now()语句调用该文件；否则，如果函数名和文件名不一致时，函数调用就需要通过文件名和与函数声明中对应的参数列表来实现。

编写好的函数 M 文件，相当于 MATLAB 提供的命令，可以在命令行进行函数调用。但要注意，要求被调用的函数对应的.m 文件必须在 MATLAB 路径下。

3.4 MATLAB 程序流程控制

和各种常见的高级语言一样，MATLAB 也提供了多种经典的程序结构控制语句。MATLAB 中的程序流程控制语句有：分支控制语句（if 结构和 switch 结构）、循环控制语句（for 循环、while 循环、continue 语句和 break 语句）和程序终止语句（return 语句），下面分别进行介绍。

1. 程序分支控制语句

分支语句可以使程序中的一段代码只在满足一定条件时才执行，因此也成为分支选择。MATLAB 中分支语句有两类：if 语句和 switch 语句。

- if 与 else 或 elseif 连用，偏向于是非选择，当某个逻辑条件满足时执行 if 后的语句，否则执行 else 语句。
- switch 和 case、otherwise 连用，偏向于情况的列举，当表达式结果为某个或某些值时，执行特定 case 指定的语句段，否则执行 otherwise 语句。

if 结构的语法形式如下所示：

```
if logical_expression
    statements
elseif logical_expression
          statements
else      logical_expression
          statements
end
```

其中 elseif 和 else 语句都是可选语句。if、elseif 和 else 构成的各项分支里面，哪个的条件满足（逻辑表达式 logical_expression 的结果为真），就执行哪个分支后面紧跟的程序语句。因此，各个分支条件满足的情况应该是互斥的和完全的，就是被选的条件在一个分

支中成立，而且只能在一个分支中成立。

当然，省略了 elseif 和 else 分支的语句，就不必要求分支条件满足的情况具备完全性了。

if 结构中条件判断除了可以用逻辑表达式外，还可以用数组 A，这时判断相当于逻辑表达式 all(A)，即当数组 A 的所有元素都为非零值时，才执行该条件后的分支代码。

特别地，当数组 A 为空数组时，相当于该条件判断为假。

switch 结构的语法形式如下所示：

```
switch expression (scalar or string)
  case value1
    statements      % Executes if expression is value1
  case value2
    statements      % Executes if expression is value2
  ...
  otherwise
    statements      % Executes if expression does not
                  % match any case
end
```

执行中，先计算表达式 expression 的值，当结果等于某个 case 的 value 时，就执行该 case 紧随的语句。如果所有 case 的 value 都和 expression 计算结果不相等，则执行 otherwise 后面紧随的语句。

otherwise 语句是可选的，当没有 otherwise 语句时，如果所有 case 的 value 都和 expression 计算结果不相等，则跳过 switch-case 语句段，直接执行后续代码。

相等的意义，对于数值类型来说，相当于判断“if result==value”，对于字符串类型来说，相当于判断“if strcmp(result,value)”。

由此可见，switch-case 语句实际上可以被 if-elseif-else 语句等效替换，不过两种结构各有更便利的地方，读者在以后的例子中会逐渐体会到。

学过 C 语言的读者需要注意，MATLAB 中的 switch-case 结构，只执行表达式结果匹配的第一个 case 分支，然后就跳出 switch-case 结构。因此，在每一个 case 语句中不需要用 break 语句跳出。

在一条 case 语句后可以列举多个值，只需要以元胞数组的形式列举多个值，就是用花括号把用逗号或空格分隔的多个值括起来即可。

2. 程序循环控制语句

循环控制语句能够使得某段程序代码多次重复执行，MATLAB 中提供了两类循环语句，分别是 for 循环和 while 循环：

- for 循环一般用在已知循环执行次数的情况；
- while 循环则用在已知循环退出条件的情况。

MATLAB 还提供了 continue 和 break 语句，用于更精细地控制循环结构：

- continue 语句使得当前次循环不向下执行，直接进入下一次循环；
- break 语句则是退出该循环。

（1）for 循环

for 循环用于知道循环次数的情况，其语法格式如下所示：

```
for index = start:increment:end
    statements
end
```

index 为循环变量，increment 为增量，end 用于判断循环是否应该终止。增量 increment 默认值为 1，可以自由设置；当增量为正数时，循环开始先将 index 赋值为 start，然后判断 index 是否小于等于 end。若是，则执行循环语句，执行完后，对 index 累加一个增量 increment；再判断 index 是否小于等于 end，若是，则继续执行循环语句，继续对 index 累加，循环往复，直到 index 大于 end 时退出循环。

增量 increment 也可以设置为负整数，表示每次循环执行后对循环变量 index 进行递减，当 index 小于 end 时，退出循环。

MATLAB 中，循环的执行效率很低，提高程序效率的一个办法就是多采用数组结构和 MATLAB 内联函数。

for 循环中的循环变量 index 也可以赋值为数组 A，那么在第一次循环中 index 就被赋值为 A(:,1)，即 A 的第一列元素，第二次循环 index 被赋值为 A(:,2)，以此类推；若 A 有 n 列元素，则循环执行 n 次，第 n 次循环时，循环变量 index 被赋值为 A(:,n)。

（2）while 循环

while 循环用于已知循环退出条件的情况，其语法形式如下所示：

```
while expression
    statements
end
```

当表达式 expression 的结果为真时，就执行循环语句，直到表达式 expression 的结果为假，才退出循环。

如果表达式 expression 是一个数组 A，则相当于判断 all(A)。特别地，空数组则被当作逻辑假，循环停止。

（3）continue 语句

continue 语句用在循环中，表示当前次循环不再继续向下执行，而是直接对循环变量进行递增，进入下一次循环。

（4）break 语句

break 语句用于退出循环。

3. 程序终止控制语句

一般程序代码都是按流程执行完毕后正常退出，但当遇到某些特殊情况，程序需要立即退出时，就可以用 return 语句提前终止程序运行。

例 3-2 return 语句使用实例。

```
clear
clc
n=-2;
if n<0
   disp('negative number!');
   return;
end
disp('codon after return')
```

本例中当变量 *n* 取值为负数时，通过 return 直接退出，不执行 if 后的代码。其运行结果是：

```
negative number!
```

若去掉其中的 return 语句，则运行结果变为：

```
negative number!
codon after return
```

return 语句更多地用在 MATLAB 函数 M 文件中。

3.5 MATLAB 中的函数及调用

3.5.1 函数类型

MATLAB 中的函数可以分为匿名函数、M 文件主函数、嵌套函数、子函数、私有函数和重载函数。

1. 匿名函数

匿名函数通常是很简单的函数，它是面向命令行代码的函数，通常只由一句很简单的声明语句组成。

匿名函数也可以接受多个输入和输出参数。使用匿名函数的优点是不需要维护一个 M 文件，而只需要一句非常简单的语句，就可以在命令窗口或 M 文件中调用函数，这对于那些函数内容非常简单的情况是很方便的。

创建匿名函数的标准格式如下所示：

```
fhandle = @(arglist) expr
```

其中，

1）“expr” 通常是一个简单的 MATLAB 变量表达式，实现函数的功能，比如 *x*+*x*.^2、sin(*x*).*cos(*x*)等；

2）“arglist” 是参数列表，它指定函数的输入参数列表，对于多个输入参数的情况，通常要用逗号分隔各个参数；

3）符号 “@” 是 MATLAB 中创建函数句柄的操作符，表示对由输入参数列表 arglist 和表达式 expr 确定的函数创建句柄，并把这个函数句柄返回给变量 fhandle，这样，以后就可以通过 fhandle 来调用定义好的这个函数。

例如定义函数：

```
myfunhd=@(x)(x+x..^2)
```

表示创建了一个匿名函数，它有一个输入参数 x，它实现的功能是 $x+x.^\wedge 2$，并把这个函数句柄保存在变量“myfunhd”中，以后就可以通过“myfunhd(a)”来计算当“$x=a$”的时候的函数值。

需要注意的是，匿名函数的参数列表 arglist 中可以包含一个参数或多个参数，这样调用的时候就要按顺序给出这些参数的实际取值。

但 arglist 也可以不包含参数，即留空，这种情况下调用函数时还是需要通过 fhandle() 的形式来调用，即要在函数句柄后紧跟一个空的括号。否则，只显示 fhandle 句柄对应的函数形式。

匿名函数可以嵌套，即在 expr 表达式中可以用函数来调用一个匿名函数句柄。

例 3-3 匿名函数创建实例。

```
>> myfhd1=@(x)(x+x.^2)
myfhd1 =     @(x)(x+x.^2)
>> myfhd1(2)
ans =      6
>> myfhd2=@(x,y)(sin(x)+cos(y))
myfhd2 =     @(x,y)(sin(x)+cos(y))
>> myfhd2(pi/2,pi/6)
ans =    1.8660
>> myfhd3=@()(3+2)
myfhd3 =     @()(3+2)
>> myfhd3()
ans =      5
>> myfhd3
myfhd3 =     @()(3+2)
>> myffhd=@(a)(quad(@(x)(a.*x.^2+1./a.*x+1./a^2),0,1)) %匿名函数嵌套使用
myffhd =     @(a)(quad(@(x)(a.*x.^2+1./a.*x+1./a^2),0,1))
>> myffhd(0.5)
ans =    5.1667
```

匿名函数可以保存在.mat 文件中，上例中就可以通过“save myfunquad.mat myffhd”把匿名函数句柄“myffhd”保存在“myfunquad.mat”文件中，以后需要用到匿名函数“myffhd”时，只需要使用语句“load myfunquad.mat myffhd”就可以了。

2. M 文件主函数

每一个函数 M 文件第一行定义的函数就是 M 文件主函数，一个 M 文件只能包含一个主函数，并通常习惯上将 M 文件名和 M 文件主函数名设为一致。

M 文件主函数的说法是针对其内部嵌套函数和子函数而言的。一个 M 文件中除了一个主函数外，还可以编写多个嵌套函数或子函数，以便在主函数功能实现中进行调用。

3. 嵌套函数

在一个函数内部，可以定义一个或多个函数，这种定义在其他函数内部的函数就被称为嵌套函数。嵌套可以多层发生，就是说一个函数内部可以嵌套多个函数，这些嵌套函数内部又可以继续嵌套其他函数。

嵌套函数的书写语法格式如下所示：

```
function x = A(p1, p2)
...
   function y = B(p3)
   ...
   end
...
end
```

一般函数代码中结尾是不需要专门标明“end”的，但在使用嵌套函数时，无论嵌套函数还是嵌套函数的父函数（直接上一层函数）都要明确标出“end”表示函数结束。

嵌套函数的互相调用需要注意和嵌套的层次密切相关，如在下面一段代码中：

```
function A(x, y)     %外层函数 A（例如主函数）
B(x, y);
D(y);
   function B(x, y)  %A 的嵌套函数（以 A 为参照可以称为第一层嵌套函数），B 的父函数为 A
   C(x);
   D(y);
      function C(x)  %B 的嵌套函数（以 A 为参照可以称为第二层嵌套函数），C 的父函数为 B
      D(x);
      end
   end
   function D(x)     %A 的嵌套函数（以 A 为参照可以称为第一层嵌套函数），D 的父函数为 A
   E(x);
      function E(x)  %D 的嵌套函数（以 A 为参照可以称为第二层嵌套函数），E 的父函数为 D
      ...
      end
   end
end
```

1）外层的函数可以调用向内一层直接嵌套的函数（A 可以调用 B 和 D），而不能调用更深层的嵌套函数（A 不可以调用 C 或 E）；

2）嵌套函数可以调用与自己具有相同父函数的其他同层嵌套函数（B 和 D 可以互相调用）；

3）嵌套函数也可以调用其父函数或与父函数具有相同父函数的其他嵌套函数（C 可以调用 B 和 D），但不能调用与其父函数具有相同父函数的其他嵌套函数内深层嵌套的函数。

4. 子函数

一个 M 文件只能包含一个主函数，但一个 M 文件中可以包含多个函数，这些编写在主函数后的函数都称为子函数。所有子函数只能被其所在 M 文件中的主函数或其他子函

数调用。

所有子函数都有自己独立的声明和帮助、注释等结构，只需要在位置上处在主函数之后即可，而各个子函数的前后顺序都可以任意放置，和被调用的前后顺序无关。

M 文件内部发生函数调用时，MATLAB 首先检查该 M 文件中是否存在相应名称的子函数，然后检查这一 M 文件所在目录的子目录下是否存在同名的私有函数，然后按照 MATLAB 路径检查是否存在同名的 M 文件或内部函数。

根据这一顺序，函数调用时首先查找相应的子函数，因此，可以通过编写同名子函数的方法实现 M 文件内部的函数重载。

子函数的帮助文件也可以通过 help 命令显示，如 myfun.m 文件中有名为 myfun 的主函数和名为 mysubfun 的子函数，那么可以通过 help myfun>mysubfun 命令来获取子函数 mysubfun 的帮助。

5. 私有函数

私有函数是具有限制性访问权限的函数，它们对应的 M 文件需要保存在名为"private"的文件夹下，这些私有函数代码编写上和普通的函数没有什么区别，也可以在一个 M 文件中编写一个主函数和多个子函数，以及嵌套函数。

但私有函数只能被 private 目录的直接父目录下的脚本 M 文件或 M 文件主函数调用。

通过 help 命令获取私有函数的帮助，也需要声明其私有特点，例如要获取私有函数 myprifun 的帮助，就要通过 help private/myprivfun 命令。

6. 重载函数

"重载"是计算机编程中非常重要的概念，它经常用在处理功能类似但参数类型或个数不同的函数编写中。

例如现在要实现一个计算功能，一种情况下输入的几个参数都是双精度浮点类型，同时也有一种情况是，输入的几个参数都是整型变量，这时候，用户就可以编写两个同名函数，一个用来处理双精度浮点类型的输入参数，另一个用来处理整型的输入参数，这样，当用户实际调用函数时，MATLAB 就会根据实际传递的变量类型选择执行其中一个函数。

MATLAB 中重载函数通常放置在不同的文件夹下，通常文件夹名称以符号@开头，然后跟一个代表 MATLAB 数据类型的字符。

例如"@double"目录下的重载函数的输入参数应该是双精度浮点型，而"@int32"目录下的重载函数的输入参数应该是 32 位整型。

3.5.2 函数参数传递

MATLAB 中通过 M 文件编写函数时，只需要指定输入和输出的形式参数列表，只是在函数实际被调用的时候，才需要把具体的数值提供给函数声明中给出的输入参数。

MATLAB 中参数传递过程是传值传递，也就是说，在函数调用过程中，MATLAB 将传入的实际变量值赋给形式参数指定的变量名，这些变量都存储在函数的变量空间中，这和工作空间变量空间是独立的，每一个函数在调用中都有自己独立的函数空间。

例如编写函数：

```
function y=myfun(x,y,z)
```

在命令窗口通过 a = myfun(3, 2, 0.5)调用此函数，那么 MATLAB 首先会建立 myfun 函数的变量空间，把 3 赋值给 x，把 2 赋值给 y，把 0.5 赋值给 z，然后执行函数实现的代码；在执行完毕后，把 myfun 函数返回的参数 y 的值传递给工作空间变量 a，调用过程结束后，函数变量空间被清除。

1．输入和输出参数的数目

MATLAB 的函数可以具有多个输入或输出参数。通常在调用时，需要给出和函数声明语句中一一对应的输入参数；而输出参数个数可以按参数列表对应指定，也可以不指定。不指定输出参数调用函数时，MATLAB 默认把输出参数列表中第一个参数的值返回给工作空间变量“ans”。

MATLAB 中可以通过 nargin 和 nargout 函数确定函数调用时实际传递的输入和输出参数个数，结合条件分支语句，就可以处理函数调用中指定不同数目的输入输出参数的情况。

例 3-4 显示函数输入和输出参数的数目实例。

```
function [y1,y2]=mytestnio(x1,x2)
if nargin==1
   y1=x1;
   if nargout==2
      y2=x1;
   end
else
   if nargout==1
      y1=x1+x2;
   else
      y1=x1;
      y2=x2;
   end
end
```

这个函数可以处理一个或两个输入参数、一个或两个输出参数的情况。当只有一个输入参数 $x1$ 和一个输出参数 $y1$ 时，把 $x1$ 赋值给 $y1$；当有 1 个输入参数 $x1$ 和两个输出参数 $y1$、$y2$ 时，把 $x1$ 赋值给 $y1$ 和 $y2$；当有两个输入参数 $x1$、$x2$ 和一个输出参数 $y1$ 时，把 $x1+x2$ 的计算结果赋值给 $y1$；当有两个输入参数 $x1$、$x2$ 和两个输出参数 $y1$、$y2$ 时，把 $x1$ 赋值给 $y1$，并把 $x2$ 赋值给 $y2$。函数调用结果如下所示：

```
>> x=mytestnio(5)
x =     5
>> [x,y]=mytestnio(5)
x =     5
y =     5
>> mytestnio(5)
ans =     5
```

```
>> x=mytestnio(5,7)
x =   12
>> [x,y]=mytestnio(5,7)
x =     5
y =     7
>> mytestnio(5,7)
ans =     5
```

指定了输入和输出参数个数的情况比较好理解，只要对应函数 M 文件中对应的 if 分支项即可；而不指定输出参数个数的调用情况，MATLAB 是按照指定了所有输出参数的调用格式对函数进行调用的，不过在输出时只是把第一个输出参数对应的变量值赋给工作空间变量 ans。

例如“mytestnio(5,7)”这句函数调用中，实际上是按照“[y1, y2]=mytestnio(x1, x2)”这种形式调用的，在函数变量空间中 x1 被赋值为 5，x2 被赋值为 7，y1 计算结果为 5，y2 计算结果为 7，但函数只把输出参数列表中第一个输出变量（即 y1）的取值返回给工作空间变量 ans，因此，ans 取值为 5。

2. 可变数目的参数传递

函数 nargin 和 nargout 结合条件分支语句，可以处理可能具有不同数目的输入和输出参数的函数调用，但这要求对每一种输入参数数目和输出参数数目的组合分别进行代码编写。

有些情况下，用户可能并不能确定具体调用中传递的输入参数或输出参数的个数，即具有可变数目的传递参数，MATLAB 中可以通过 varargin 和 varargout 函数实现可变数目的参数传递，使用这两个函数对于处理具有复杂的输入输出参数个数组合的情况也是便利的。

函数 varargin 和 varargout 把实际的函数调用时传递的参数值封装成一个元胞数组，因此，在函数实现部分的代码编写中，就要用访问元胞数组的方法访问封装在 varargin 或 varargout 中的元胞或元胞内的变量。

例 3-5 可变数目的参数传递实例。

```
function y=mytestvario(varargin)
temp=0;
for i=1:length(varargin)
    temp=temp+mean(varargin{i}(:));
end
y=temp/length(varargin);
```

本例中的函数 mytestvario 以 varargin 为输入参数，从而可以接受可变数目的输入参数。函数实现部分首先计算了各个输入参数（可能是标量、一维数组或二维数组）的均值，然后计算这些均值的均值。调用结果如下所示：

```
>> mytestvario(4)
ans =     4
>> mytestvario(4,[1 3])
ans =     3
>> mytestvario(4,[1 3],[1 23;23 1],magic(4))
ans =    6.6250
```

对于"mytestvario(4,[1 3],[1 23;23 1],magic(4))"这句函数调用，在函数变量区，varargin首先被赋值为一个元胞数组"{4,[1 3],[1 23;23 1],magic(4)}"，即 varargin 有 1 行 4 列个元胞，各个元胞中分别存储了一个标量数值、一维行数组、2 行 2 列的二维数组和 4 行 4 列的魔方数组；在函数实现部分，首先创建中间变量 temp，并初始化赋值为零（用来存储各个元胞中数据均值的总和），然后计算每一个元胞中所有数据的均值并将结果累加到 temp上；最后通过"y=temp/length(varargin)"计算这些均值的均值。

函数 varargin 和 varargout 也可以放置在参数列表中某些必然出现的参数之后，其语法格式有如下几种形式。

1）function [out1, out2] = example1(*a*, *b*, varargin)，表示函数 example1 可以接受大于等于两个输入参数，返回两个输出参数；两个必选的输入参数是 *a* 和 *b*，其他更多的输入参数被封装在 varargin 中。

2）function [i, j, varargout] = example2(*x*, *y*)，表示函数 example2 接受两个输入参数 *x* 和 *y*，返回大于等于两个输出参数，前两个输出参数为 *i* 和 *j*，其他更多的输出参数封装在 varargin 中。

函数 varargout 和 varargin 的用法类似，只需要注意访问时应按照访问元胞数组的方法，这里就不再举例了。

3. 返回被修改的输入参数

MATLAB 函数有独立于 MATLAB 工作空间的自己的变量空间，因此，输入参数在函数内部的修改都只具有和函数变量空间相同的生命期，如果不指定将修改后的输入参数值返回到工作空间，那么在函数调用结束后这些修改后的值将被自动清除。

例 3-6 函数内部的输入参数修改实例。

```
function y=mytest(x)
x=x+5;
y=x*2;
```

本例中的 mytest 函数内部，首先修改了输入参数 *x* 的值（*x*=*x*+5），然后以修改后的 *x* 的值计算输出参数 *y* 的值（*y*=*x**2）。调用结果如下所示：

```
>> x=3
x =     3
>> y=mytest(x)
y =    16
>> x
x =     3
```

由此结果可见，调用结束后，函数变量区中的 *x* 在函数调用中被修改，但此修改只在函数变量区有效，这并没有影响到 MATLAB 工作空间变量空间中的变量 *x* 的值。函数调用前后，MATLAB 工作空间中的变量 *x* 始终取值为 3。

那么，如果用户希望函数内部对输入参数的修改也对 MATLAB 工作空间的变量有效，就需要在函数输出参数列表中返回此输入参数。

对于本例中的函数，则需要把函数修改为“function [y,x]=mytest(x)”，而在调用时也要通过“[y,x]= mytest(x)”这种形式。

例 3-7 函数参数传递实例。将修改后的输入参数返回给 MATLAB 工作空间。

```
function [y,x]=mynewtest(x)
x=x+5;
y=x*2;
```

MATLAB 工作空间中的调用结果如下所示：

```
>> x=3
x =     3
>> [y,x]=mynewtest(x)
y =    16
x =     8
>> x
x =     8
```

通过本例可见，函数调用后，MATLAB 工作空间中的变量 x 取值从 3 变为 8（3+5），可见通过[y,x]=mynewtest(x)调用，实现了函数对 MATLAB 工作空间变量的修改。

4. 全局变量

通过返回修改后的输入参数，可以实现函数内部对 MATLAB 工作空间变量的修改。而另一种殊途同归的方法则是使用全局变量。声明全局变量需要用到 global 关键词，语法格式为“global variable”。

通过全局变量可以实现 MATLAB 工作空间变量空间和多个函数的函数空间的共享，这样，多个使用全局变量的函数和 MATLAB 工作空间共同维护这一全局变量，任何一处对全局变量的修改，都会直接改变此全局变量的取值。

在应用全局变量时，通常在各个函数内部通过 global variable 语句声明，在命令窗口或脚本 M 文件中也要先通过 global 声明，然后进行赋值和调用。

例 3-8 全局变量使用实例。

```
function y=myprocess(x)
global T
T=T*2;
y=exp(T)*sin(x);
```

在命令窗口中声明全局变量然后赋值调用：

```
>> global T
>> T=0.3
T =    0.3000
>> myprocess(pi/2)
ans =    1.8221
>> exp(T)*sin(pi/2)
ans =    1.8221
>> T
```

```
T =    0.6000
```

通过本例可见，用 global 将 T 声明为全局变量后，函数内部对 T 的修改也会直接作用到 MATLAB 工作空间中。函数 myprocess 调用一次后，T 的值从 0.3 变为 0.6（0.3*2）。

3.6 函数句柄

函数句柄实际上提供了一种间接调用函数的方法。创建函数句柄需要用到操作符@。前面已经讲过，匿名函数实际上就是一种函数句柄，而对 MATLAB 提供的各种 M 文件函数和内部函数，也都可以创建函数句柄，从而可以通过函数句柄对这些函数实现间接调用。

函数句柄的优点如下：

（1）方便地实现函数间互相调用；

（2）兼容函数加载的所有方式；

（3）拓宽子函数，包括局部函数的使用范围；

（4）提高函数调用的可靠性；

（5）减少程序设计中的冗余；

（6）提高重复执行的效率。

创建函数句柄的一般语法格式如下所示：

```
fhandle=@function_fileame
```

其中，

"funciont_filename" 是函数所对应的 M 文件的名称或 MATLAB 内部函数的名称；

"@" 是句柄创建操作符；

"fhandle" 变量保存这一函数句柄。

例如 fhandle=@sin 就创建了 MATLAB 内部函数 sin 的句柄，并将其保存在 fhandle 变量中，以后就可以通过 fhandle(x)来实现 sin(x)的功能。

通过函数句柄调用函数时，也需要指定函数的输入参数，比如可以通过 fhandle(arg1, arg2, ..., arg$n$)这样的调用格式来调用具有多个输入参数的函数。对于那些没有输入参数的函数，在使用句柄调用时，要在句柄变量后加上空的圆括号，即 fhandle()。

例 3-9 函数句柄创建和调用实例。

```
>> fhd=@sin
fhd =    @sin
>> x=0:0.25*pi:2*pi;
>> fhd(x)
ans = 0    0.7071    1.0000    0.7071    0.0000   -0.7071   -1.0000   -0.7071
-0.0000
```

MATLAB 中提供了丰富的处理函数句柄的函数，如表 3.1 所示。

表 3.1　处理函数句柄的函数

函　　数	说　　明
functions(fhandle)	返回一个结构体，存储了函数的名称，函数类型（简单函数或重载函数），以及函数 M 文件的位置
func2str(fhandle)	将函数句柄转换为函数名称字符串
str2func(str)	将字符串代表的函数转换为函数句柄
save filename.mat fhandle	将函数句柄保存在.mat 文件中
load filename.mat fhandle	把.mat 文件中存储的函数句柄装载到工作空间
isa(var, 'function_handle')	检测变量 var 是不是函数句柄
isequal(fhda,fhdb)	检测两个函数句柄是否对应于同一个函数
feval(fhandle)	调用函数句柄 fhandle

例 3-10　处理函数句柄的函数使用实例。

```
>> fhda=@exp
fhda =     @exp
>> fhdb=@myprocess
fhdb =     @myprocess
>> functions(fhdb)
ans =  function: 'myprocess'
       type: 'simple'
       file: 'D:\MATLAB71\work\MATLABbook\EX-10\myprocess.m'
>> isa(fhda,'function_handle')
ans =     1
>> isequal(fhda,fhdb)
ans =     0
```

3.7 MATLAB 程序调试

MATLAB 程序出错主要为以下两类：

- 格式错误，如缺‘（’或‘）’等，在运行时可检测出大多数该类错误，并指出错在哪一行。
- 算法错误，逻辑上的错误，不易查找，遇到此类错误时需耐心。一般可考虑如下方法：
  - 删除句尾分号，显示中间结果；
  - 在适当位置加上 keyboard 语句，使程序暂停；
  - 在函数定义行之前加上%，注释掉，使之变成脚本语言；
  - 使用 MATLAB 调试器，设置断点，或单步执行，使用一些调试和分析工具。

下面讲述程序调试的一些工具及调试方法，熟练掌握并运用这些工具及调试方法，能提高编程的效率。

3.7.1 调试方法

MATLAB 程序有直接调试法和工具调试法这两种调试方法。

（1）直接调试法

直接调试法就是在 M 文件中，将某些语句后面的分号去掉，迫使 M 文件输出一些中间计算结构，以便发现可能的错误。常用的做法有：

1）在适当位置，添加显示某些关键变量值的语句；

2）利用 echo 指令，使运行时在屏幕上逐行显示文件内容，echo on 能显示 M 脚本文件；echo FunName On 能显示名为 FunName 的 M 函数文件；

3）在原 M 脚本或函数文件的适当位置，添加指令 keyboard，keyboard 语句可以设置程序的断点；

4）通过将原 M 函数文件的函数声明行注释掉，可使一个中间变量难于观察的 M 函数文件变为一个所有变量都保存在基本工作空间中的 M 脚本文件。

（2）工具调试法

工具调试法就是在程序中设置一些断点，利用调试菜单（Debug）中的一些选项进行调试。

Debug 菜单用于程序调试，需要与 Breakpoints 菜单项配合使用。MATLAB 7 的 Debug 菜单中的菜单项介绍如下。

- Open M-Files when Debugging：用于调试时打开 M 文件。
- Step：用于单步调试程序。
- Step In：用于单步调试进入子函数。
- Step Out：用于单步调试从子函数跳出。
- Continue：程序执行到下一断点。
- Clear Breakpoints in All Files：清除所有打开文件中的断点。
- Stop if Errors/Warnings：在程序出错或报警处停止往下执行。
- Exit Debug Mode：退出调试模式。

除了采用调试器调试程序外，MATLAB 还提供了一些命令用于程序调试。命令的功能和调试器菜单命令类似。MATLAB 提供的调试命令介绍如下。

- 快捷键 F10：实现单步调试。
- 快捷键 F11：用于单步调试进入子函数。
- 快捷键 Shift+F11：用于单步调试从子函数跳出。
- 快捷键 F5：实现程序执行到下一断点。

MATLAB 提供了进行代码调试和代码分析优化的工具，这些工具，一般的 MATLAB 用户都应该有所了解。尤其是断点调试部分的内容，建议读者尽量以自己的程序代码为例，多加练习，熟练掌握。

3.7.2 调试工具

当完成 MATLAB 代码编写后，用户就可以在命令窗口中运行代码（脚本或函数文件）。对于比较简单的代码，一般只要编程习惯较好，都可以一次通过。但对于很多比较复杂的

情况，或者用户初学 MATLAB 编程，一些常见的错误还不能避免，就容易在运行时出现错误。这时候，就需要利用 MATLAB 的调试工具对出现错误的代码进行调试纠错。

MATLAB 的代码编辑调试器是一个综合了代码编写、调试的集成开发环境。MATLAB 代码调试过程，主要是通过 MATLAB 代码编辑-调试器的 Debug 菜单下的子项进行的，如图 3-3 所示。

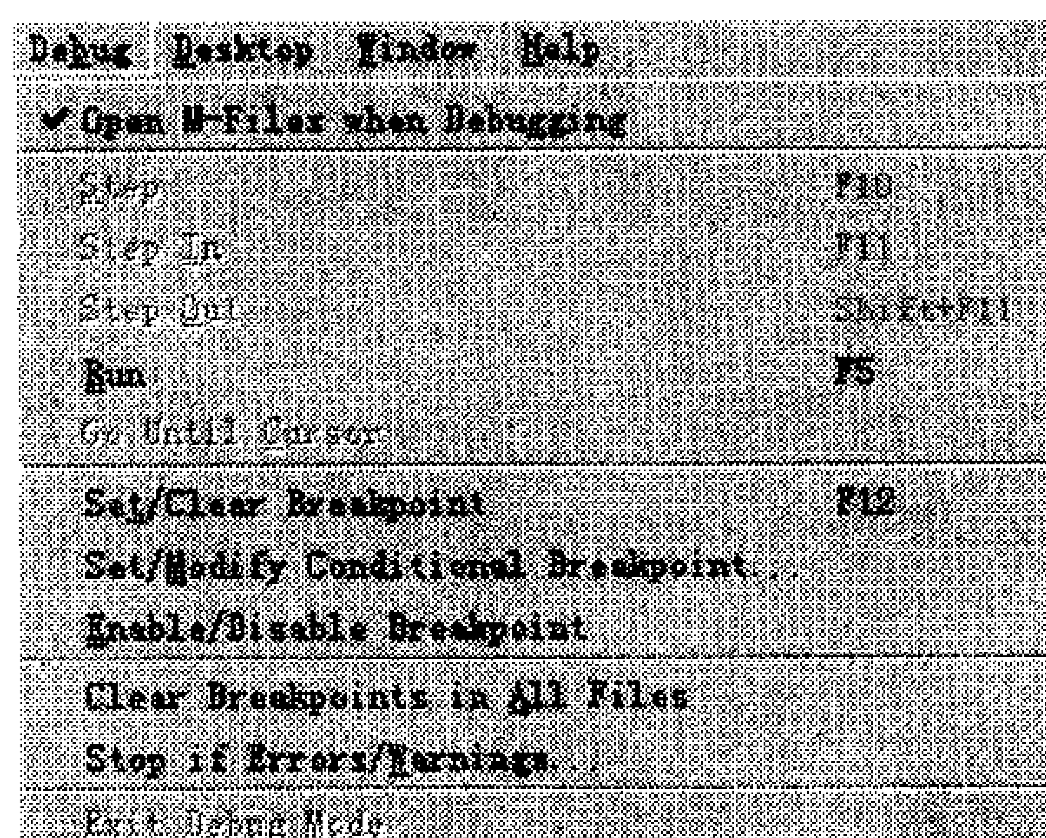


图 3-3　MATLAB 代码编辑-调试器的 Debug 菜单

Debug 菜单下各子项的含义介绍如下。

1）Step：在调试模式下，执行 M 文件的当前行，对应的快捷键是 F10。

2）Step In：在调试模式下，执行 M 文件的当前行，如果 M 文件当前行调用了另一个函数，那么进入该函数内部，对应的快捷键是 F11。

3）Step Out：当在调试模式下执行 Step In 进入某个函数内部之后，执行 Step Out 可以完成函数剩余部分的所有代码，并退出函数，暂停在进入函数内部前的 M 文件所在行末尾。

4）Run：运行当前 M 文件，快捷键是 F5；当前 M 文件设置了断点时，运行到断点处暂停。

5）Go Until Cursor：运行当前 M 文件到在光标所在行的行尾。

需要注意，以上这些调试项，除了 Run（运行），都需要首先在 M 文件中设置断点，然后运行到断点位置后，这些调试项才可启用。

6）Set/Clear Breakpoint：在光标所在行开头设置或清除断点。

7）Set/Modify Conditional Breakpoint...：在光标所在行开头设置或修改条件断点，选择此子项，会打开“条件断点设置”对话框，如图 3-4 所示，用于设置在满足什么条件时，此处断点有效。

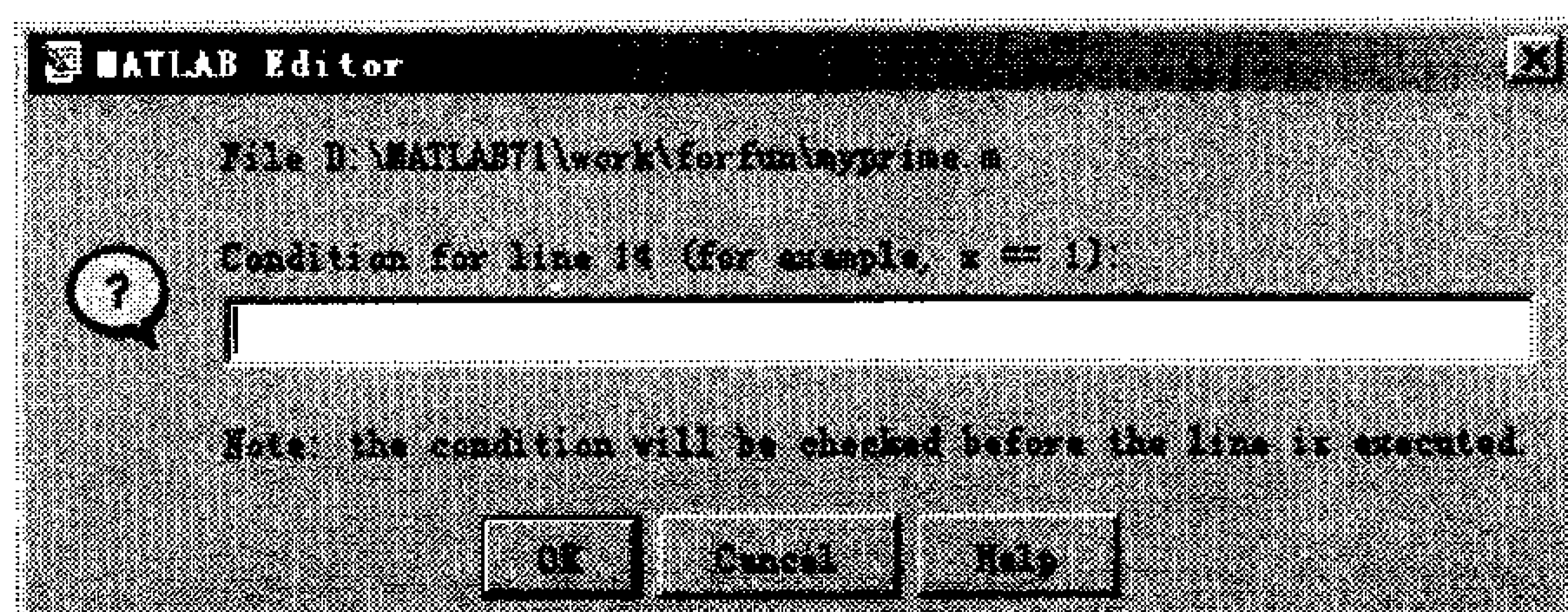


图 3-4　“条件断点设置”对话框

8）Enable/Disable Breakpoint：将当前行的断点设置为有效或无效。

9）Clear Breakpoints in All Files：清除所有 M 文件中的断点。

10）Stop if Errors/Warnings…：设置出现某种运行错误或警告时，停止程序运行，选择此子项，会打开“错误/警告设置”对话框，如图 3-5。

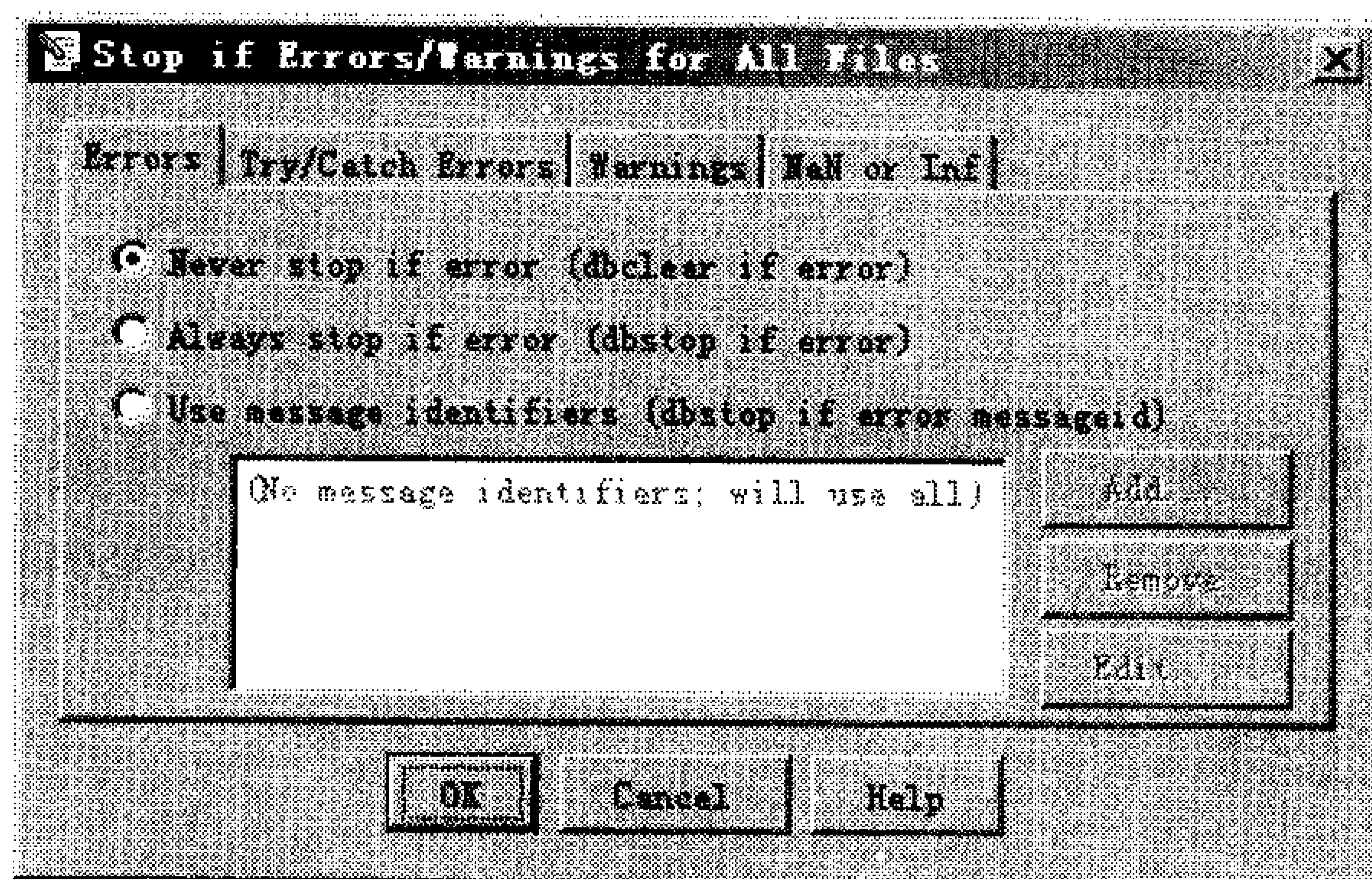


图 3-5 设置出现某种运行错误或警告则停止程序运行

11）Exit Debug Mode：退出调试模式。

上面逐项讲述了 Debug 菜单下每一个子项的意义，实际上，很多子项都有对应的快捷工具按钮。MATLAB 代码编辑-调试器中，如图 3-6 所示的部分工具按钮就是用于 M 文件调试的。

图 3-6 中的各个工具按钮，从左向右依次对应于 Set/Clear Breakpoint、Clear Breakpoints in All Files、Step、Step In、Step Out、Run、Exit Debug Mode 等菜单子项。



图 3-6 调试工具按钮

通常的调试过程是：先运行（Run）一遍 M 文件，针对具体的出错信息，在适当的地方设置断点或条件断点；再次运行（Run）到断点位置（如图 3-7 所示），此时 MATLAB 把运行控制权交给键盘，命令窗口出现“K>>”提示符（如图 3-8 所示）；此时可以在命令窗口中查询 M 文件运行过程中的所有变量，包括函数运行时的中间变量。运行到断点位置后，用户可以选择“Step/Step Into/Step Out”等调试运行方式，逐行运行并适时查询变量取值，从而逐渐找到错误所在并排除。

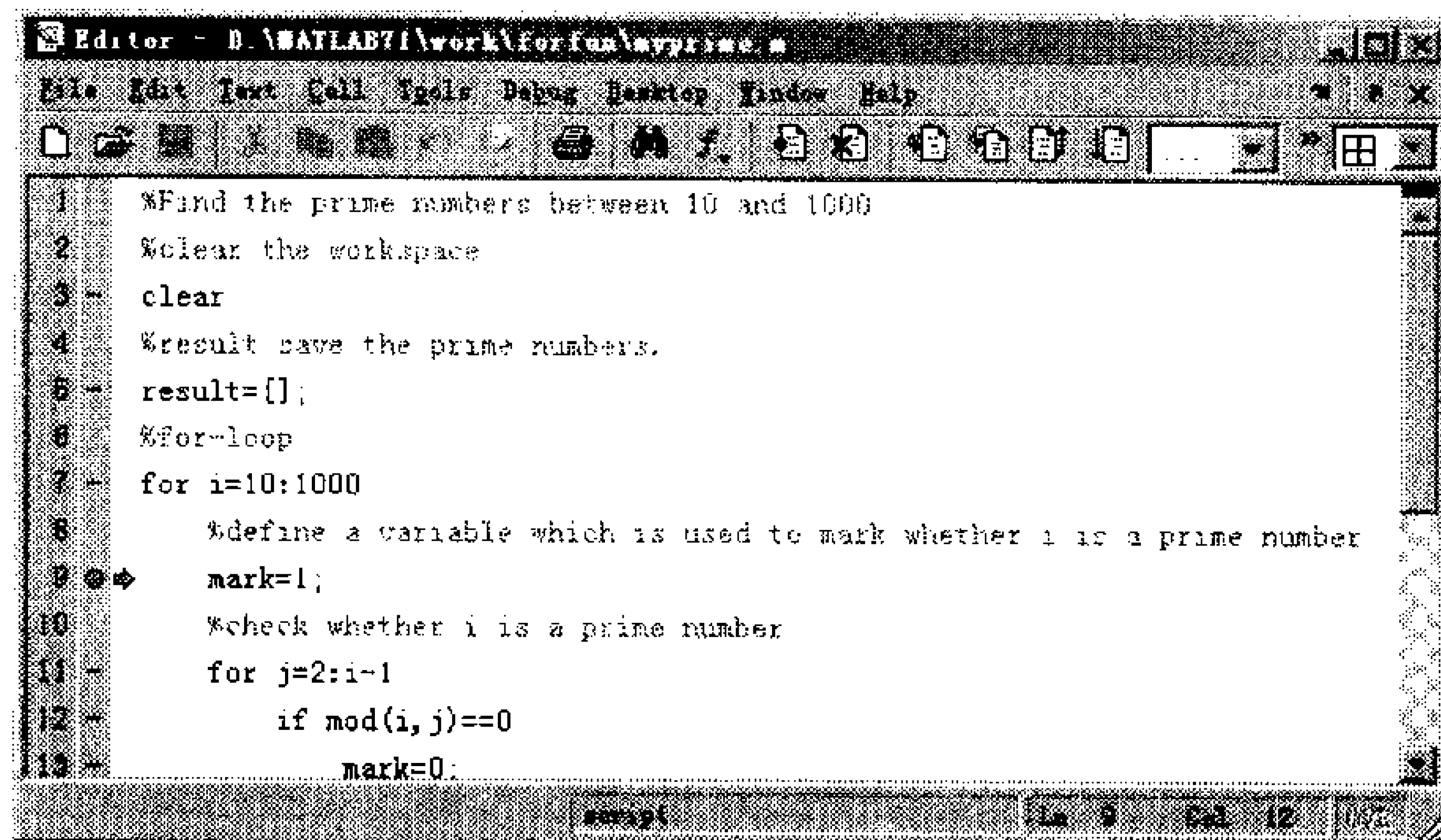


图 3-7 设置断点后运行（Run）到断点所在位置

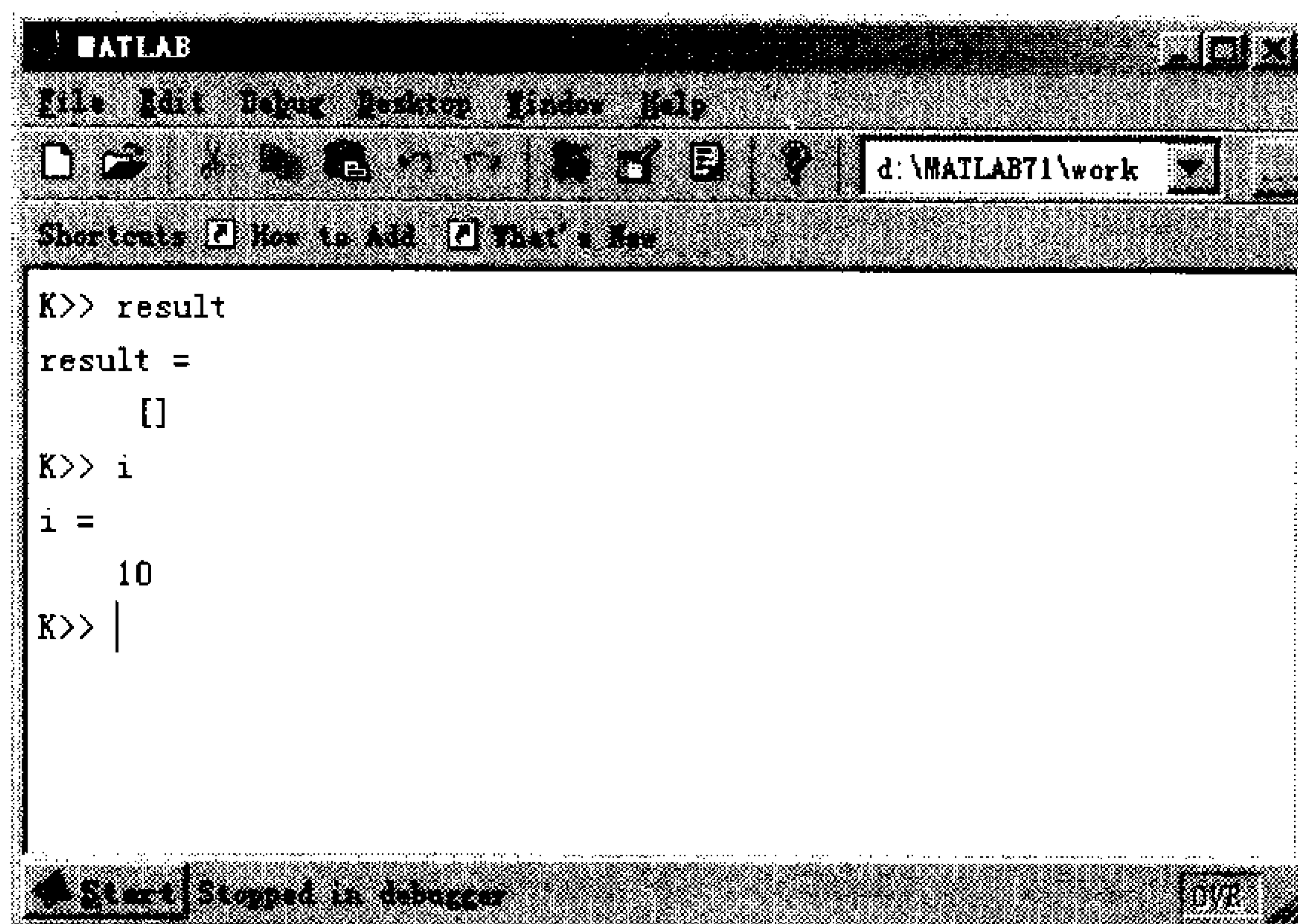


图 3-8　调试模式时 MATLAB 命令窗口把控制权交给键盘

3.7.3　M 文件分析工具

通过 Debug 项对 M 文件进行调试的过程，可以对文件中的编写错误和运行错误进行纠正。完成了调试过程后，用户编写的 M 文件就可以正确地运行了。但可能运行效率还不是最优，这就需要通过 MATLAB 提供的分析工具对代码进行分析，然后针对性地进行优化。

MATLAB 提供的 M 文件分析工具包括 M-Lint 工具和 Profiler 工具，它们都有图形操作界面，使用简单方便，是 MATLAB 程序分析优化的必用工具。

M-Lint 分析工具

M-Lint 工具可以分析用户 M 文件中的错误或性能问题。用户可以先在代码编辑-调试器中打开待分析的 M 文件，然后选择 Tools 菜单下的 Check Code with M-Lint 项，如图 3-9 所示。

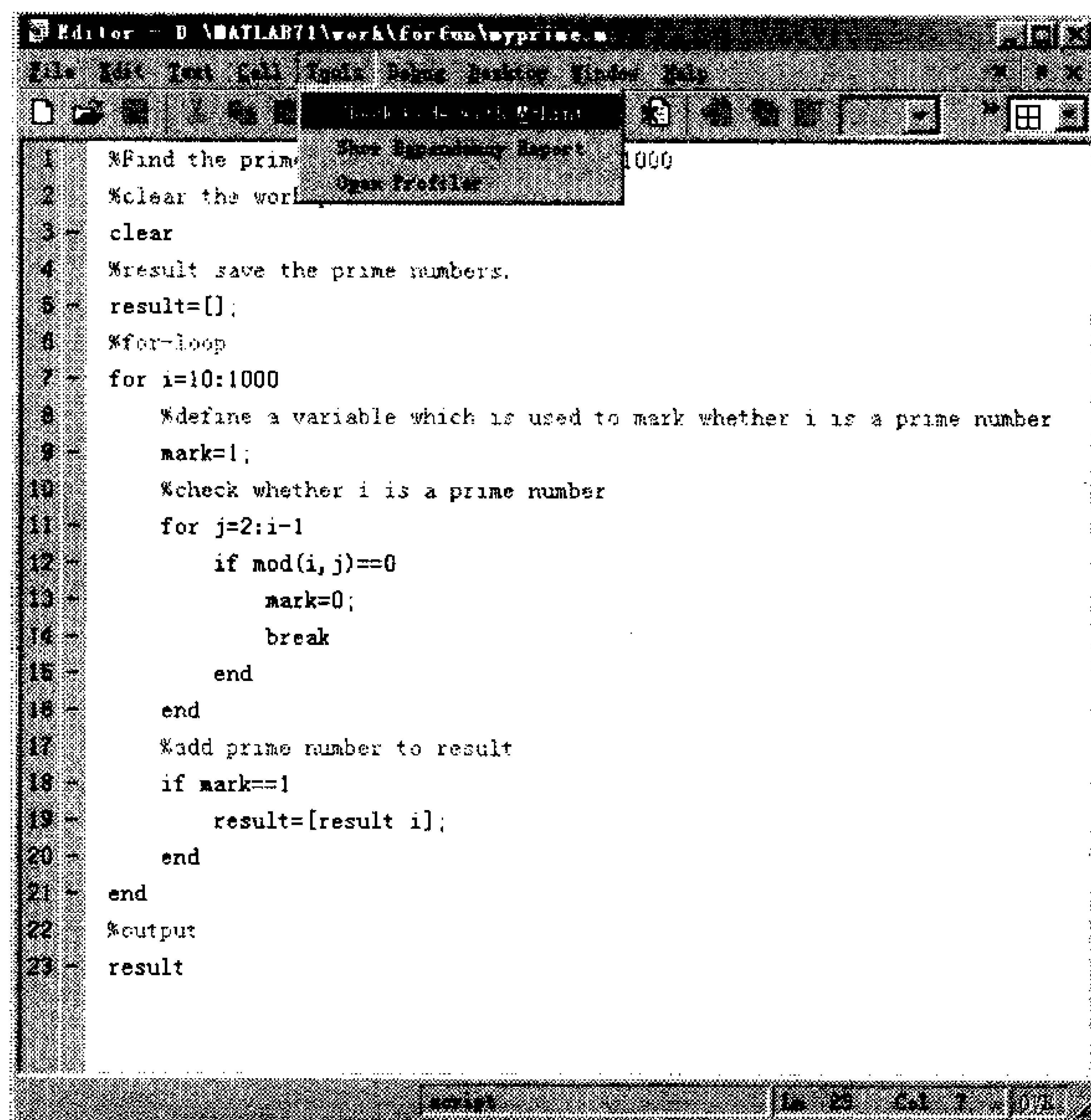


图 3-9　通过 Tools 菜单打开 M-Lint 工具

图 3-9 所示中的代码是查找 10～1000 之内所有素数的 M 文件，它的脚本如下所示。

```
%Find the prime numbers between 10 and 1000
%clear the workspace
clear
%result save the prime numbers.
result=[];
%for-loop
for i=10:1000
    %define a variable which is used to mark whether i is a prime number
    mark=1;
    %check whether i is a prime number
    for j=2:i-1
        if mod(i,j)==0
            mark=0;
            break
        end
    end
    %add prime number to result
    if mark==1
        result=[result i];
    end
end
%output
result
```

运行 M-Lint 工具后结果如图 3-10 所示。

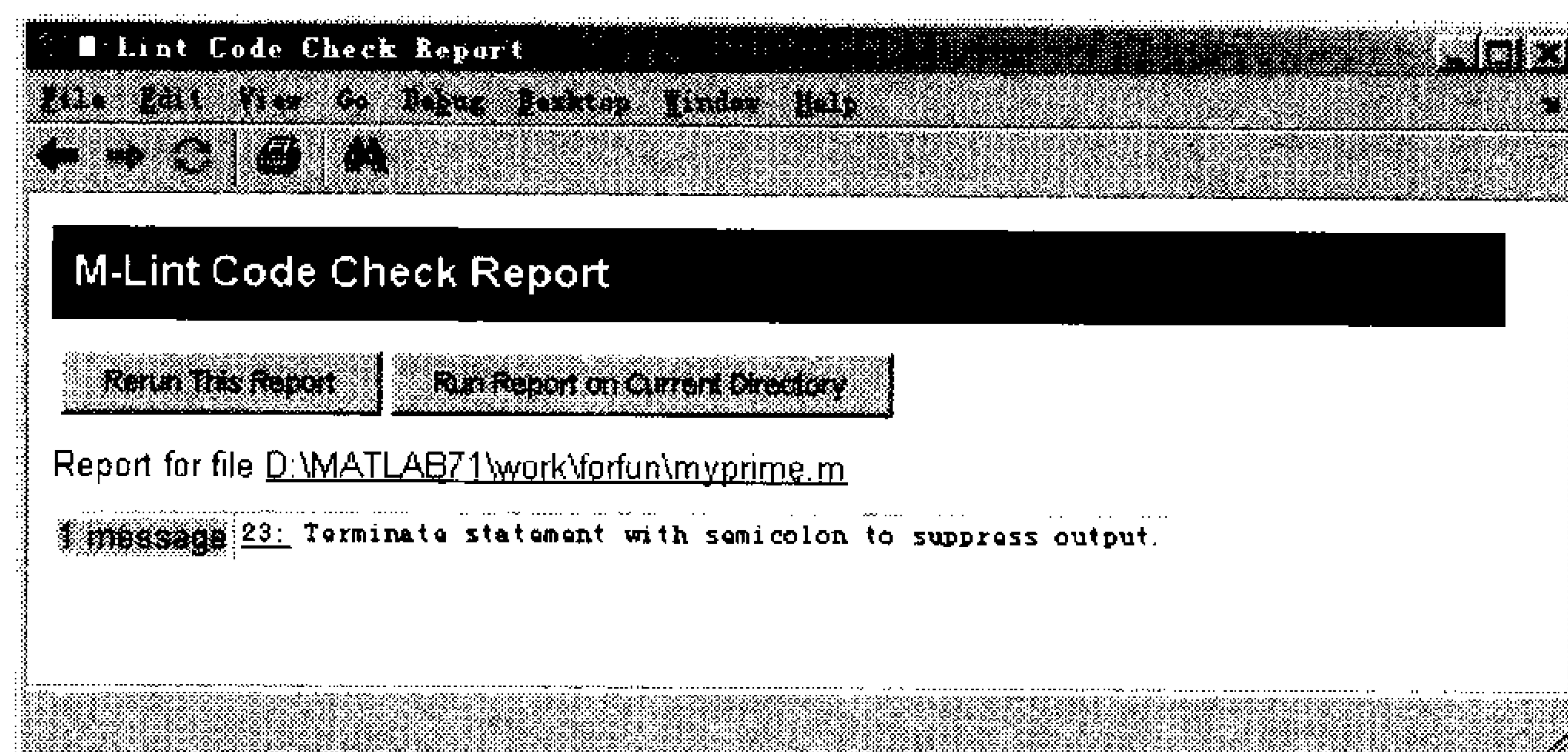


图 3-10　M-Lint 分析结果

从图 3-10 可以看出，M-Lint 分析完成后，会返回一个浏览器界面下的分析报告（Check Report），报告中包括被分析的 M 文件的路径，以及若干个分析结果（如图 3-10 所示中的 1 message 表示只有一条分析结果）。分析结果的格式是“行号：错误或问题报告”。

M-Lint 分析结果中经常出现的错误或问题报告包括：没有以分号结束以阻止中间变量输出，变量在文件中从没有被其他语句调用，以及循环过程中数组尺寸会增加等。

实际上，M-Lint 分析得到的问题报告，并不一定必须消除该问题，而是要具体问题具体分析。当用户认可某一条分析结果时，可以点击分析结果中的行号，就可以快捷打开相应的 M 文件并定位到该行，用户就可以方便地修改代码了。

M-Lint 不仅可以分析单个 M 文件，还可以分析一个文件夹下的所有 M 文件。通常在 MATLAB 主界面下，选择“Desktop”菜单下的“Current Directory”，则可以显示文件夹面板，通过单击此面板顶部的“M-Lint”工具则可以分析相应文件夹下的所有 M 文件，如图 3-11 所示。

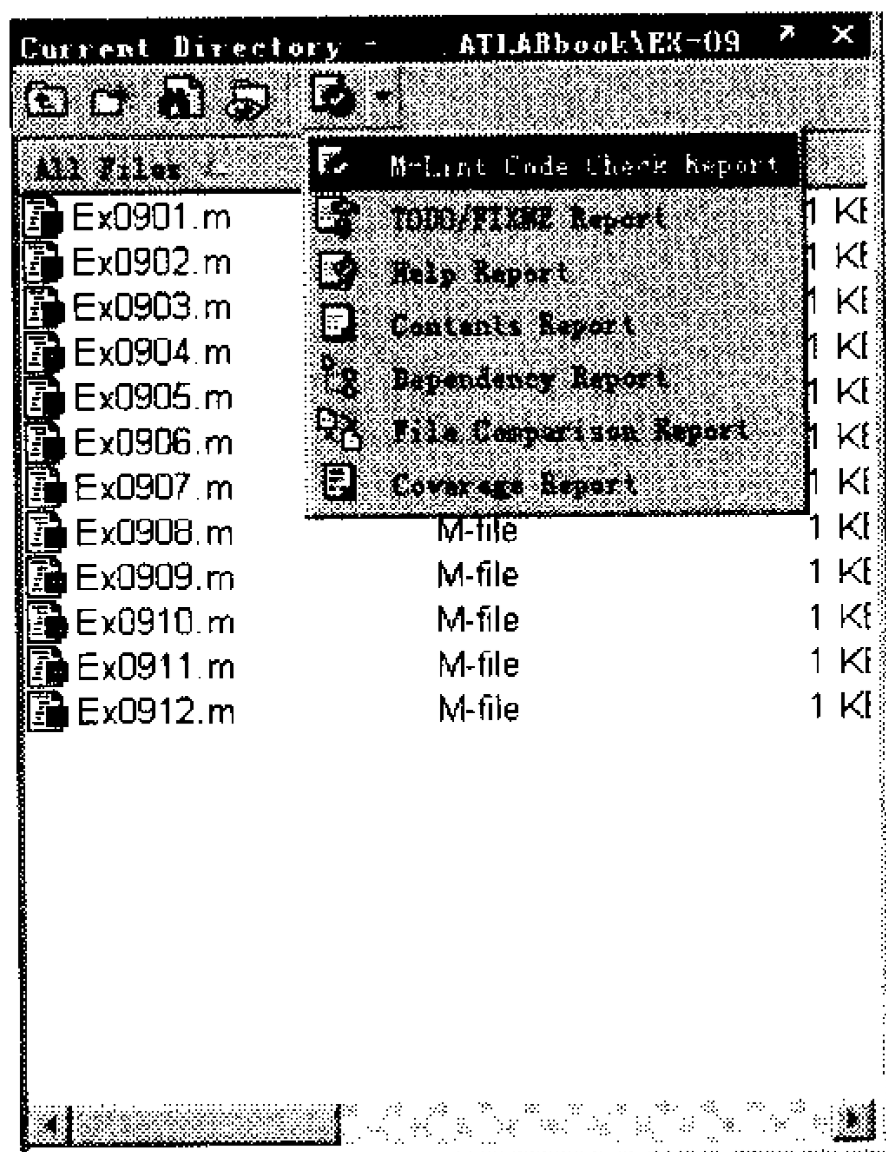


图 3-11　当前目录面板和 M-Lint 按钮

3.7.4　Profiler 分析工具

Profiler 工具是 MATLAB 提供的另一个功能强大的代码分析工具。使用时，用户可以提前在代码编辑调试器中打开 M 文件，然后选择“Tools”菜单下的“Open Profiler”项，就可以运行 Profiler 分析工具，Profiler 图形界面如图 3-12 所示。

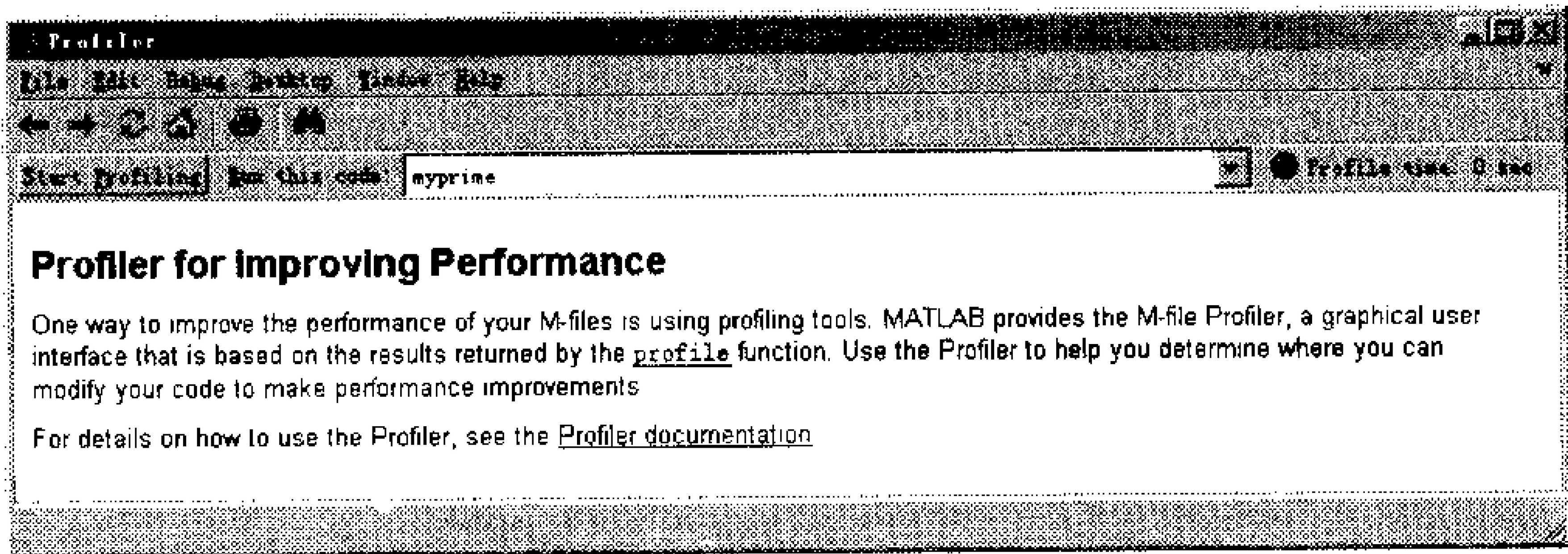


图 3-12　Profiler 工具图形界面

单击图 3-12 中所示的“Start Profiling”按钮，就可以分析此 M 文件，分析结果如图 3-13 所示。

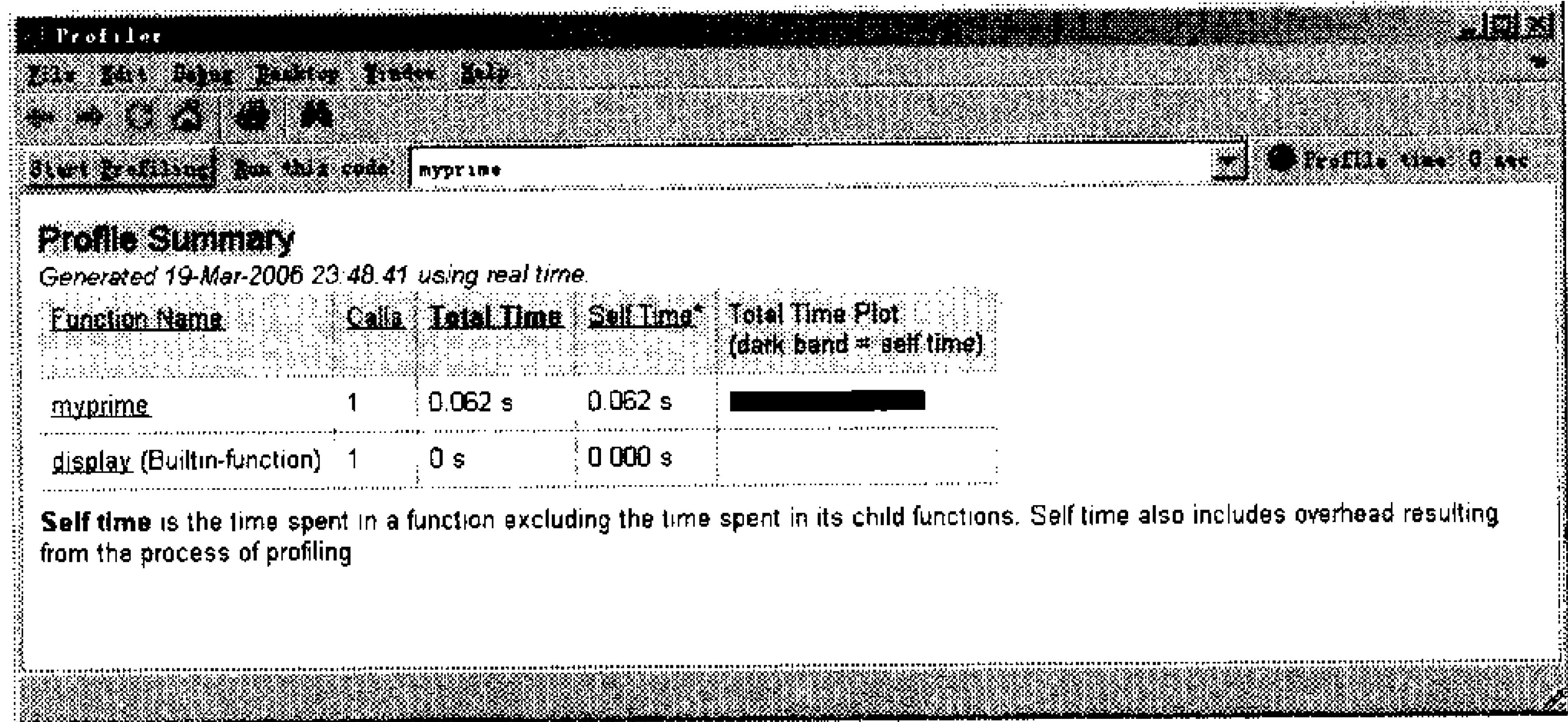


图 3-13　Profiler 分析结果

从图 3-13 可见，Profiler 分析结果给出了调用函数名称、调用次数、消耗总时间等信息。单击图 3-13 中蓝色的 “myprime”，可以打开关于此 M 文件的更加详细的分析报告，如图 3-14 所示。

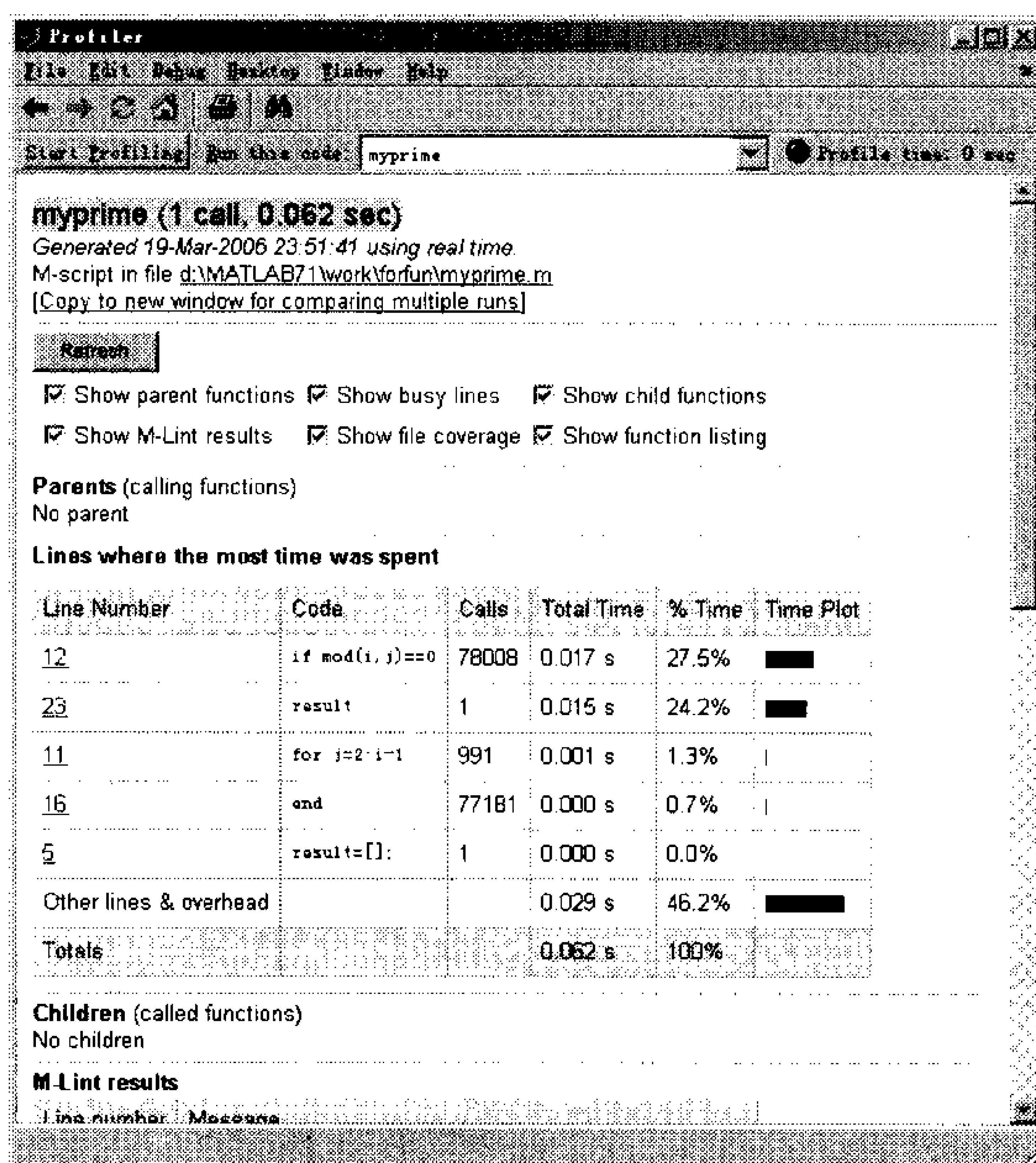


图 3-14　更详细的 Profiler 分析结果

图 3-14 所示的更详细的 Profiler 分析结果中显示了 myprime.m 文件运行中最消耗时间的部分及其具体耗时信息。用户可以有针对性地修改那些最消耗时间的部分。

一般来说，应该尽量避免不必要的变量输出、循环赋值前预定义数组尺寸，多采用向量化的 MATLAB 函数，少采用数组，这些都能够提高 MATLAB 程序的运行性能。

3.8 MATLAB 程序设计技巧

MATLAB 是一种科学计算语言，但同时也具有和 C、FORTRAN 等高级语言相类似的语言特征，能方便地实现程序控制。利用 MATLAB 的程序控制功能，可以将有关 MATLAB 命令编成程序存储在一个文件中（M 文件），然后运行该文件，MATLAB 就会自动依次执行文件中的命令，直到全部命令执行完毕。

编程时，首先要考虑到变量初值或者变量类型改变时，程序的应对能力，即程序的鲁棒性，因此出色的程序要具有较好的例外处理机制。此外，还要考虑程序的执行效率。

一些编程技巧能提高程序的执行效率，因此，掌握一些 MATLAB 的编程技巧也是非常必要的，下面对常用的 MATLAB 编程技巧进行介绍。

3.8.1　嵌套计算

一个程序的执行速度取决于它所调用的子程序个数以及采用的算法。通常希望子程序

越少越好，算法效率越高越好。

嵌套计算是一种具有较小时间复杂度的算法，如【例 3-11】所示。

例 3-11 嵌套计算与直接求值的比较实例。考虑下面两个多项式，其中式（**）为式（*）的嵌套表达式：

$$p(x)=a_3x^3+a_2x^2+a_1x+a_0 \qquad (*)$$

$$p1(x)=((a_3x+a_2)x+a_1)x+a_0 \qquad (**)$$

易知式（*）和式（**）所需要的乘法数分别为：6 次和 3 次。显然后者具有更高的执行效率。

下面用具体程序进行说明，创建函数 ex0311()。

```
function ex0311()
N = 100000;
a = [1:N];
x = 1;
tic                                     %初始化时钟
p1 = sum(a.*x.^[N-1:-1:0]);             %按照(*)求值
p1, toc                                 %时钟停止，获得执行时间
tic, p2=a(1);
for i = 2:N                             %嵌套计算(**)
    p2 = p2*x + a(i);
end
p2, toc
tic, p3 = polyval(a, x), toc            %用 MATLAB 自带函数求值
```

运行函数，输出结果如下所示：

```
p1 =  5.0001e+009
Elapsed time is 0.035852 seconds.
p2 =  5.0001e+009
Elapsed time is 0.005543 seconds.
p3 =  5.0001e+009
Elapsed time is 0.088367 seconds.
```

可见，嵌套算法耗时最少。而用 polyval 求值执行速度最慢。

采用嵌套算法不仅能够提高执行效率，而且此方法具有更强的解决问题的能力，下面的实例可以说明这个问题。

例 3-12 嵌套计算与非嵌套计算的比较实例。求 possion 分布的有限项和，形如：

$$S(M)=\sum_{n=0}^{M}\frac{\lambda^n}{n!}e^{-\lambda}$$

由概率论知识可知，当 M 很大时，上式的值趋近于 1。

分别用式（*）和式（**）来求解，创建函数 ex0312()。

```
function ex0312()
r = 80;
```

```
M = 160;
p = exp(-r);
S1 = 0;
for k = 1:M
    p=p*r/k;                    %嵌套,式(**)
    S1=S1+p;
end
S1
S2= 0;
for k = 1:M
    p = r^k/factorial(k);       %非嵌套,式(*)
    S2 = S2 + p;
end
S2*exp(-r);
S2
```

运行函数，输出结果如下所示：

```
S1 =    1.0000
S2 =  5.5406e+034
```

由结果可知，嵌套方法的结果非常接近真实值，而非嵌套的方法根本无法得到正确的结果。

3.8.2 循环计算

循环计算可按照给定的条件，重复执行指定的语句。MATLAB 用于实现循环计算的语句有 for 语句和 while 语句。

对于循环的使用需要注意以下几点。

（1）尽量避免使用循环。在 MATLAB 编程中，采用循环会降低程序的执行速度，应尽量避免使用，可以用其他方式，如向量运算等代替。

（2）为了得到最大的速度，在 for 循环被执行之前，应预先分配数组。否则，在 for 循环内每执行一次命令，对数组重新分配一次内存，这样会降低 MATLAB 的执行效率。

（3）优先考虑内联（inline）函数，矩阵运算应该尽量采用 MATLAB 的内联函数，因为内联函数是由更底层的编程语言 C 语言构造的，其执行速度显然快于使用循环的矩阵运算。

（4）应用 MEX 技术。尽管采用了很多措施，但执行速度仍然很慢，比如说耗时的循环是不可避免的，这样就应该考虑用其他语言，如 C 或 FORTRAN 语言。按照 MEX 技术要求的格式编写相应部分的程序，然后通过编译连接，形成在 MATLAB 中可以直接调用的动态链接库（DLL）文件，这样可以显著地加快运算速度。有关 MEX 技术及其应用的详细内容可参考相关书籍。

3.8.3 使用例外处理机制

优秀的程序员能够指导用户如何使用他编写的程序，而且在用户使用不当时，能够给

出错误提示信息，并引导用户正确使用函数。前面编写的一些程序都没有相应的错误处理机制，如函数 ex0313，其输入应大于零，当输入小于零时会得到错误的结果。【例 3-13】给出了这样的实例。

例 3-13 例外处理机制使用实例。用户输入错误的函数参数时的例外处理，给函数 ex0313 输入错误的参数，如下所示：

```
>> ex0313(-1)
Elapsed time is 0.000017 seconds.
m =     1
```

显然错误的结果是由于用户不清楚函数参数的取值范围导致的。这种错误由于并不影响程序的运行，因而很难被发现。所以程序员在编程时，应当考虑到可能发生的此类错误，并给出处理错误的机制和错误提示信息。

对上例，如果用户输入了错误的参数，可以采用下面语句终止程序，并提示出错。

```
if n<0
   error('input must be positive, stopped');
end
```

此时，再执行上面的命令的结果如下所示：

```
>> ex0313 (-1)
??? Error using ==> ex0313
input must be positive, stopped
```

这种错误多数都是由于越界造成的，尤其在使用矩阵时，要注意引用矩阵位置不要超过它的边界。

另外，如果用户输入的函数参量超过设定的最大个数，或者类型不合要求也会出现这种错误。

但一般而言，对于输入参量小于设定个数的情形，MATLAB 内置程序一般会对未赋值参量作默认处理。典型的例子是 plot(*Y*, *X*)函数，plot(*Y*)默认的 *X* 坐标是[0, 1, 2…]序列。

程序员在编写程序的时候也应当注意处理这种情况。采用 nargin 函数可以判断输入参量的个数，从而设定未被指定的输入参数的值或者直接报错。

例 3-14 nargin 函数应用实例。利用 nargin 函数，实现两个多项式的相加，并具有一定的报错功能。

编写函数 ex0314。

```
function p=ex0314(a, b)      %求多项式 a, b 和 p
if nargin==1                  %输入参数个数为 1，另一个默认参数为全 0 向量
   b=zeros(4, 1);
elseif nargin==0
   error('empty input');      %输入参数个数为 0，报错
end
a=a(:).' ;b=b(:).' ;
```

```
na=length(a) ; nb=length(b) ;
p=[zeros(1, nb-na) a]+[zeros(1, na-nb) b];  %多项式相加
```

在 MATLAB 命令窗口中输入正确的参数，调用该函数，输出正确的结果，如下所示：

```
>> a=[1 2 3 4];
>> ex0314(a)
ans =     1     2     3     4
```

在 MATLAB 命令窗口中输入错误的参数，调用该函数，输出报错结果，如下所示：

```
>> ex0314()
??? Error using ==> ex0314
empty input
```

可见，用户输入参数不合要求时，会作默认处理或报错，使得程序具有更强的适应性。

3.8.4 使用全局变量

全局变量是指在不同的工作空间以及基本的工作空间中可以共享的变量。用户只需要在主程序或者任何子程序中声明一个或多个全局变量， 则函数和主程序中都可以直接引用它们，采用如下格式生成全局变量。

```
global v1 v2…vn
```

表达式中各变量之间用空格隔开。

使用全局变量时要注意以下几点。

（1）它可以在主程序和函数之间不需要经过输入或输出变量直接传递数据。但要注意在函数调用中使用它们时，调用结束后全局变量在工作空间中仍然存在。

（2）两个或多个函数也可以共有一个全局变量，只要同时在这些函数中用 global 语句加以定义即可。

（3）使用全局变量时必须十分小心，最好把全局变量名取得长一些或全部用大写，以免与函数中的局部变量重名。如果重名，容易出现致命错误。所以，使用全局变量不是一个好的编程方法。

（4）一旦变量被声明为全局的，则在任何声明它的地方都可以对它进行修改。这在一定程度上破坏了子程序的独立性。如果全局变量被多个子程序修改，则用户很难知道全局变量的确切值，这使得程序的可读性大大下降。

下面用实例进行说明全局变量的用法。

例 3-15 全局变量使用实例。本例用以说明全局变量的声明及函数传递。

建立子程序 ex0315.m 和主程序 ex0315main.m，同时在子程序 ex0315.m 以及主程序 ex0315main.m 中定义全局变量 *D*，具体程序如下所示：

```
function x = ex0315(t, D)
global D                          %声明全局变量
t(find(t == 0)) = eps;            %防止分母出现 0 项
```

```
x = sin(pi*t/D)./(pi*t/D);
function ex0315main()%主函数
global D
D = 2;
b1 = -2;
b2 = 2;
t = b1 +[0:100]/100*(b2 - b1);
%通过全局变量传递参数
plot(t, ex0315(t))
```

本程序运行结果如图 3-15 所示。

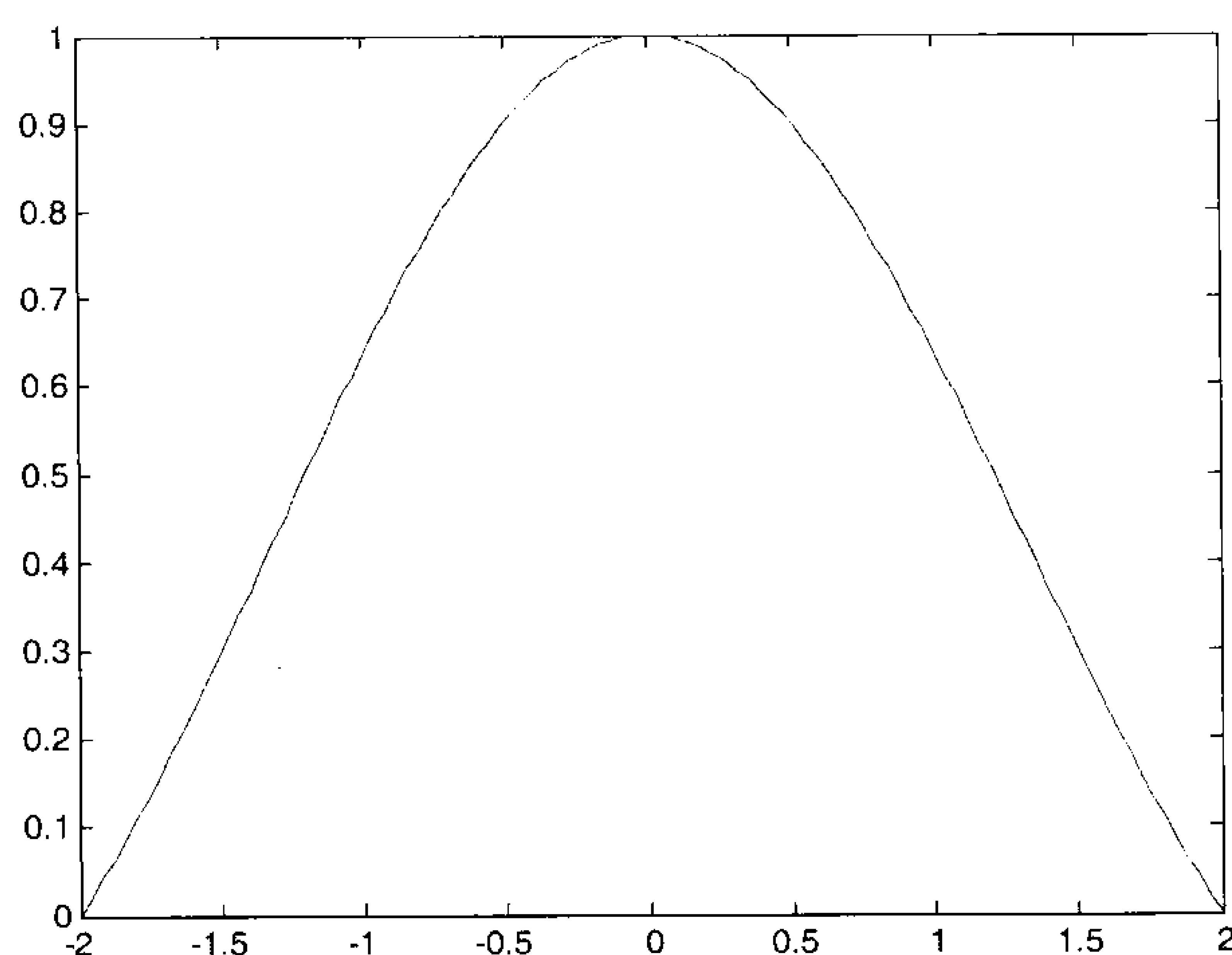


图 3-15 【例 3-15】输出结果

如果在子程序 ex0315 中修改全局变量的值，则变量声明时即对其进行赋值。例如将 ex0315 中的声明语句改为：global *D*=1，则运行后可得图 3-16 所示结果。

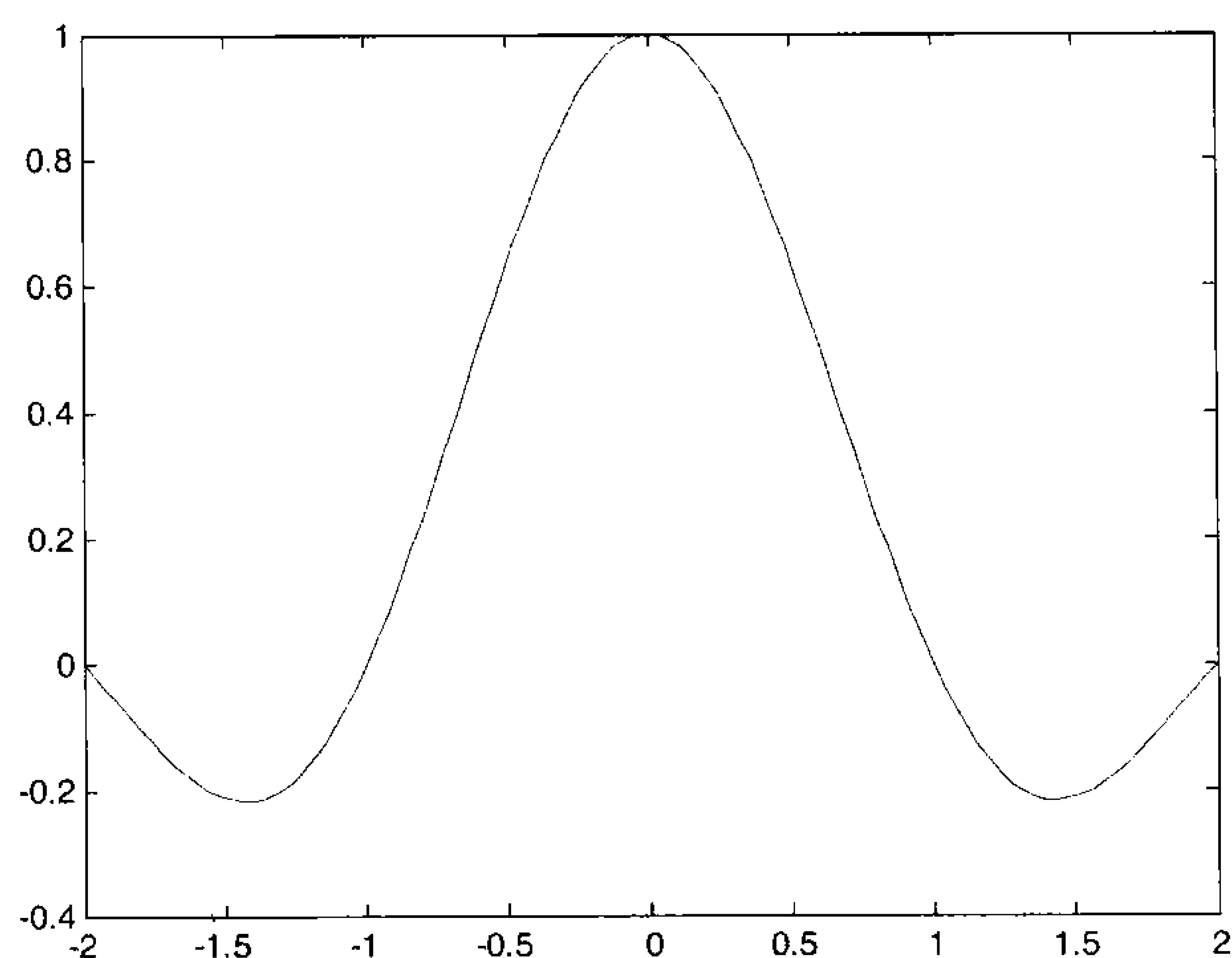


图 3-16 【例 3-15】中全局变量在子程序中被修改的输出结果

显然，ex0315 中对全局变量的赋值覆盖了 ex0315main 中的赋值，但这种修改在主程序中却不能得到反映。如果主程序和子程序不是同一个程序员编写，几乎很难找到结果与

程序不符合的原因。不过，在使用全局变量时，MATLAB 会给出如下的警告信息，提示用户注意全局变量是否在其他地方被修改：

Warning: The value of local variables may have been changed to match the globals. Future versions of MATLAB will require that you declare a variable to be global before you use that variable.

3.8.5 通过 varargin 传递参数

在编写函数时 varargin 只能作为函数的最后一个参量，主要传递函数中调用的子函数中可选项的参数，其大小也随着输入参量的变化而发生改变。

例 3-16 通过 varargin 传递参数的实例。该实例实现在不同输入参数时对函数 ex0315 的作图，其中通过 varargin 传递可选参数。

分别编写主程序 ex0316.m 和作图程序 ex0316plot.m。

```
function ex0316()  %主程序。通过 varargin 传递可选参数
D = 1; b1 = -2; b2 = 2;
t = b1+[0:100]/100*(b2 - b1);
bounds = [b1 b2];
subplot(1,3,1),  ex0316plot('ex0315')   %作出 x 的函数图，bouds 为默认值[-1,1]
axis([b1 b2 -0.4 1.2])
subplot(1,3,2),  ex0316plot('ex0315',bounds)%输入两参数，作[-2,2]上 x 函数图
axis([b1 b2 -0.4 1.2])
subplot(1,3,3), ex0316plot('ex0315',bounds,D)  %可选项输入为 1
axis([b1 b2 -0.4 1.2])
function ex0316plot(ftn,bounds,varargin)
                                          %子程序。varagin 为可选变量，输入时可以不考虑
if nargin < 2
   bounds = [-1 1];                                    %bounds 默认为[-1,1]
end
b1 = bounds(1); b2 = bounds(2);
t = b1 + [0:100]/100*(b2 - b1);
x = feval(ftn, t, varargin{:});
plot(t,x)
```

运行主程序，输出结果如图 3-17 所示。

对于一个具有多输入参量的程序，传统的方法是针对所有可能出现的输入情况进行处理。上例对应了 3 种输入参量情况，即输入参量个数分别为 1、2、3 的情况。普通处理方法是利用 if-elseif-else 结构对其进行处理。当变量很多时，该方法过于烦琐。

由于 varargin 本身可以根据输入参数个数的变化自动调节大小，因此，在调用具有多个可选参数的子程序的过程中使用时，可以大大简化程序。

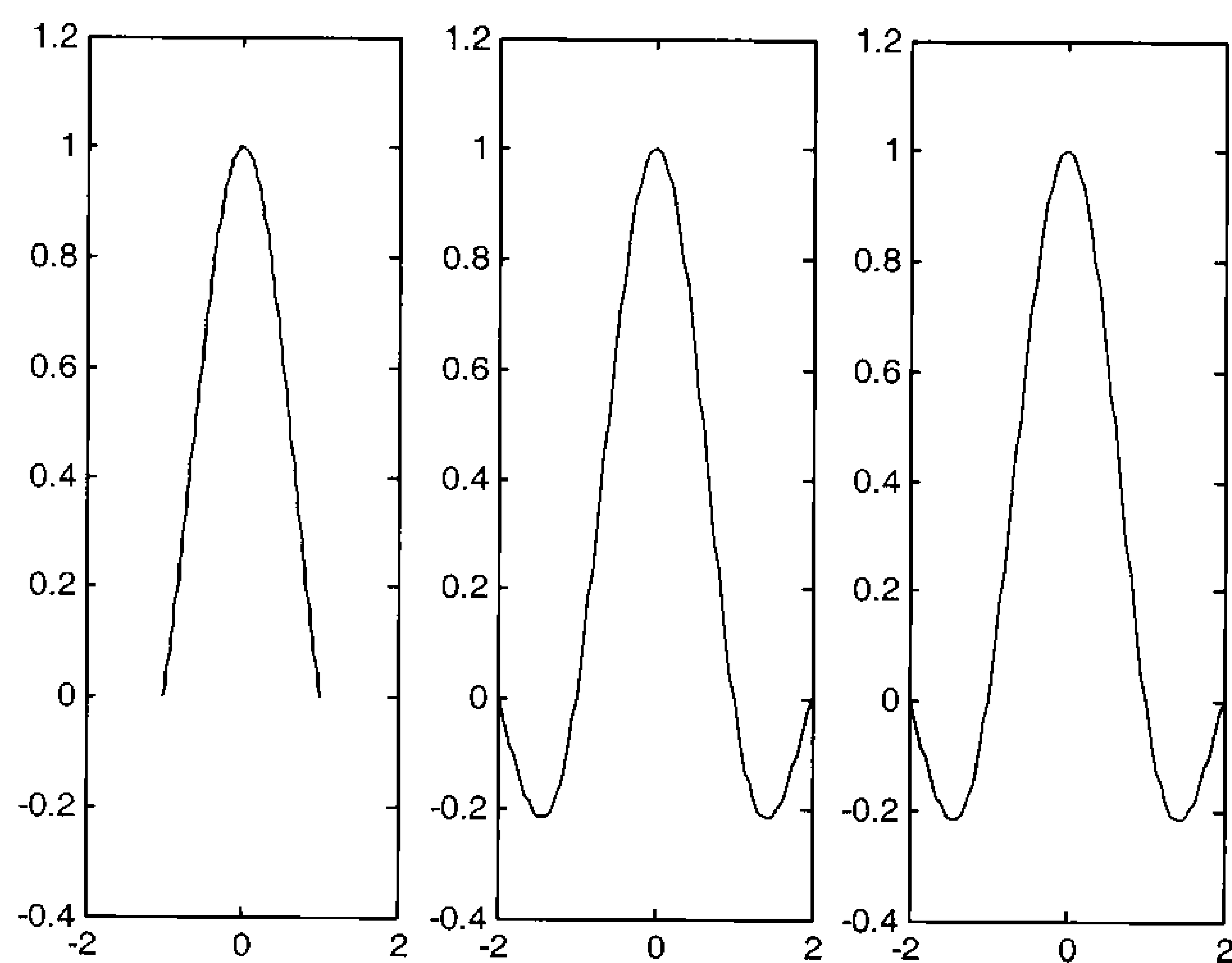


图 3-17 varargin 函数传递参数

3.9 小结

本章围绕数值计算介绍了 MATLAB 程序设计的基础知识，包括 MATLAB 的基本操作和编程技巧，这些都是后面内容的基础。

MATLAB 拥有众多的内置函数，学习 MATLAB 时，读者不要试图完全记住或者掌握它们，需要学会使用 help、lookfor 等命令查找所需的命令或函数。

在编写 MATLAB 程序，尤其是大型、复杂的 MATLAB 程序时，要多从用户角度考虑，力求让程序的例外处理机制完美，具有更好的可读性，但同时也要考虑算法的执行效率，找到这两方面的一个较好的平衡。

Part 2

下篇　算法程序篇

第 4 章　插值

插值是函数逼近的一种重要方法，是数值计算的基本课题。插值是一种求函数近似值的方法，针对某个插值点 x，用插值节点 x_i 上的已知值 $f(x_i)$ 组合生成 $f(x)$ 的近似值。最简单的插值法是多项式插值法，本章介绍的几种插值法中，基本上是以多项式插值法为主，只不过它们的构造方法各自不同，从而它们的误差和代数精度也不同。

通过本章，读者不仅能掌握常见的插值算法，而且还能熟练使用 MATLAB 编程来实现这些算法。

4.1 拉格朗日插值

拉格朗日插值法是基于基函数的插值方法，插值多项式可表示为：

$$L(x)=\sum_{i=0}^{n} y_i l_i(x)$$

其中 $l_i(x)$ 称为 i 次基函数：

$$l_i(x)=\frac{(x-x_0)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}$$

则一般离散数据 $(x_i, y_i), i=0,1,2,\cdots,n$ 的拉格朗日插值多项式如下：

$$L(x)=\sum_{i=0}^{n} y_i l_i(x)=\sum_{i=0}^{n} y_i \frac{(x-x_0)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}$$

很容易验证它满足插值条件。

在 MATLAB 中编程实现的拉格朗日插值法函数为：Language

功能：求已知数据点的拉格朗日插值多项式

调用格式：[f,f0] = Language (x,y,x0)

其中，x：　已知数据点的 x 坐标向量；

y：　已知数据点的 y 坐标向量；

x0：　插值点的 x 坐标；

f：　求得的拉格朗日插值多项式；

f0：　x0 处的插值。

在 MATLAB 中实现拉格朗日插值的代码如下所示：

```
function f = Language(x,y,x0)
%求已知数据点的拉格朗日插值多项式
%已知数据点的 x 坐标向量：x
%已知数据点的 y 坐标向量：y
%插值点的 x 坐标：x0
%求得的拉格朗日插值多项式：f
% x0 处的插值：f0
syms t;
if(length(x) == length(y))
    n = length(x);
else
    disp('x 和 y 的维数不相等！');
    return;
end                                          %检错
f = 0.0;
for(i = 1:n)
    l = y(i);
    for(j = 1:i-1)
        l = l*(t-x(j))/(x(i)-x(j));
    end;
    for(j = i+1:n)
        l = l*(t-x(j))/(x(i)-x(j));          %计算拉格朗日基函数
    end;
    f = f + l;                               %计算拉格朗日插值函数
    simplify(f);                             %化简
end
f0 = subs(f,'t',x0);                         %计算插值点的函数值
```

例 4-1　拉格朗日插值法应用实例。根据下面的数据点求出其拉格朗日插值多项式，并计算当 $x=1.6$ 时 y 的值。

x	0	0.5	1.0	1.5	2.0	2.5	3.0
y	0	0.4794	0.8415	0.9975	0.9093	0.5985	0.1411

解：在 MATLAB 命令窗口中输入以下命令：

```
>> x=0:0.5:3;
>> y=[ 0    0.4794    0.8415    0.9975    0.9093    0.5985    0.1411];
>> [f,f0]=Language(x,y,1.6)              %计算输出的拉格朗日插值多项式
f =-5397851692524447/2111062325329920*t*(t-1)*(t-3/2)*(t-2)*(t-5/2)*(t-3)+
3789648413623927/3377699720527872*t*(t-1/2)*(t-3/2)*(t-2)*(t-5/2)*(t-3)-8984
636099947915/5066549580791808*t*(t-1/2)*(t-1)*(t-2)*(t-5/2)*(t-3)+4095111552
621091/3377699720527872*t*(t-1/2)*(t-1)*(t-3/2)*(t-5/2)*(t-3)-67381973129455
/211106232532992*t*(t-1/2)*(t-1)*(t-3/2)*(t-2)*(t-3)+1016876825140703/810647
93292668928*t*(t-1/2)*(t-1)*(t-3/2)*(t-2)*(t-5/2)
```

```
%计算 x=1.6 时的插值输出值 f0
f0 =    0.9996
```

表格中的数据点是按 $y=\sin x$ 给出的，sin(1.6)=0.9996，从插值函数在 x=1.6 的值看出，插值函数的精度是比较高的。

正如某些实验数据的情况，如果一组固定的节点 x_i 对应多组不同的数值 y_i，此时用拉格朗日插值比较方便，因为每一种情况下基函数都保持不变，但是一旦对插值问题添加新的节点，那么所有的基函数都得重新计算，这时就不适合用拉格朗日插值，后面介绍的均差型的牛顿插值就比较有用了。

4.2 艾特肯插值

拉格朗日插值法简单且易于建立，但它有一个缺点，那就是一旦增加插值节点的个数，原有的多项式的计算结果并不能加以利用，插值多项式必须重新建立，而且其形式不易简化，计算比较复杂。

艾特肯插值法是通过递推来建立插值多项式的，因此比较容易利用递推公式来建立插值公式。艾特肯插值法的基本思想是 $k+1$ 次插值多项式可由两个 k 次插值多项式通过线性插值得到。艾特肯插值法的推导如下。

令

$$p_{0,1,\cdots,k,n}(x)=\frac{1}{x_n-x_k}\begin{vmatrix}p_{0,1,\cdots,k}(x) & x_k-x\\ p_{0,1,\cdots,k-1,n}(x) & x_n-x\end{vmatrix}$$

利用上式，可得到

$$p_{0,1}(x)=\frac{1}{x_1-x_0}\begin{vmatrix}p_0(x) & x_0-x\\ p_1(x) & x_1-x\end{vmatrix}$$

$$p_{0,2}(x)=\frac{1}{x_2-x_0}\begin{vmatrix}p_0(x) & x_0-x\\ p_2(x) & x_2-x\end{vmatrix}$$

递推下去，可得到下面的计算表格（$p_{0,1,\cdots,k,n}(x)$ 代表多项式，它下标中的 n 代表有 $n+1$ 个插值点，k 代表插值多项式是 $k+1$ 次的）：系数的计算过程如表 4-1 所示。

表 4-1　艾特肯插值多项式计算表格

已知数据	构造过程
x_0　$p_0(x)=f(x_0)$	
x_1　$p_1(x)=f(x_1)$	$p_{0,1}(x)$
x_2　$p_2(x)=f(x_2)$	$p_{0,2}(x)$　$p_{0,1,2}(x)$
x_3　$p_3(x)=f(x_3)$	$p_{0,3}(x)$　$p_{0,1,3}(x)$　$p_{0,1,2,3}(x)$
…	…
x_n　$p_n(x)=f(x_n)$	$p_{0,n}(x)$　$p_{0,1,n}(x)$　$p_{0,1,2,n}(x)$　…　$p_{0,1,2,\cdots,n}(x)$



说明：表右边每一项的计算公式分析如下，设第 i 行第 j 列的多项式为 $A(i,j)$，其中 $i=1,2,\cdots,n$， $j=1,2,\cdots,n$。

例如， $A(1,1)=p_{0,1}(x)$， $A(3,2)=p_{0,1,3}(x)$

则： $A(i,j)=A(i,j-1)\dfrac{x-x_{j-1}}{x_i-x_{j-1}}-A(j-1,j-1)\dfrac{x-x_i}{x_i-x_{j-1}}$

最右下角的 $p_{0,1,\cdots,n-1,n}(x)$ 就是最终的插值多项式。

> 在 MATLAB 中编程实现的艾特肯插值法函数为：Atken
>
> 功能：求已知数据点的艾特肯插值多项式
>
> 调用格式：[f ,f0]= Atken (x,y,x0)
>
> 其中，x：　已知数据点的 x 坐标向量；
>
> y：　已知数据点的 y 坐标向量；
>
> x0：　插值点的 x 坐标；
>
> f：　求得的艾特肯插值多项式；
>
> f0：　x0 处的插值。

在 MATLAB 中实现艾特肯插值的代码如下所示：

```
function [f,f0] = Atken(x,y,x0)
%求已知数据点的艾特肯插值多项式
%已知数据点的 x 坐标向量：x
%已知数据点的 y 坐标向量：y
%插值点的 x 坐标：x0
%求得的艾特肯插值多项式：f
% x0 处的插值：f0
syms t;
if(length(x) == length(y))
    n = length(x);
else
    disp('x 和 y 的维数不相等！');
    return;
end                                        %检错
y1(1:n) = t;                               %符号函数数组赋初值
for(i=1:n-1)
    for(j=i+1:n)
        y1(j) = y(j)*(t-x(i))/(x(j)-x(i))+y(i)*(t-x(j))/(x(i)-x(j));
    end
    y = y1;
    simplify(y1);
end
f = y1(n);
f0 = subs(y1(n),'t',x0);                   %计算插值点的函数值
```

例 4-2　艾特肯插值法应用实例。根据下面的数据点求出其艾特肯插值多项式，并计算当 $x=1.6$ 时 y 的值。

x	1	1.2	1.8	2.5	4
y	0.8415	0.9320	0.9738	0.5985	−0.7568

解：在 MATLAB 命令窗口中输入以下命令：

```
>> x=[1 1.2 1.8 2.5 4];
>> y=[0.8415 0.9320 0.9738 0.5985 -0.7568];
>> [f,f0]=Atken(x,y)  %计算输出的艾特肯插值多项式
f =2/3*(5/11*(5/14*(-15983/30000*t+10307/7500)*(t-6/5)-5/14*(181/400*t+
389/1000)*(t-4))*(t-9/5)-5/11*(5/3*(1323/8000*t+5409/8000)*(t-6/5)-5/3*(181/
400*t+389/1000)*(t-9/5))*(t-4))*(t-5/2)-2/3*(10/7*(10/13*(-81/500*t+2007/2000)*
(t-6/5)-10/13*(181/400*t+389/1000)*(t-5/2))*(t-9/5)-10/7*(5/3*(1323/8000*t+
5409/8000)*(t-6/5)-5/3*(181/400*t+389/1000)*(t-9/5))*(t-5/2))*(t-4)
%计算 x=1.6 时的插值输出值 f0
f0 =    0.9992
```

表格中的数据点是按 $y=\sin x$ 给出的，$\sin(1.6)=0.9996$，从插值函数在 $x=1.6$ 的值看出，插值函数的精度是比较高的。

艾特肯插值可以根据精度的要求逐步提高插值的阶，在插值过程中只需逐步将两个低阶的插值结果进行线性组合即可，因而很方便。

如果无须求出插值多项式，只需求出插值点的值，艾特肯插值的优点就体现出来了：即如果在计算过程中，存在$\left|p_{0,1,2,\cdots,k+1}(x_0)-p_{0,1,2,\cdots,k}(x_0)\right|<\varepsilon$，其中$\varepsilon$为给定精度值，则计算可到此为止，从而节省了计算量。

4.3 利用均差的牛顿插值

函数 f 的零阶均差定义为 $f[x_0]=f(x_0)$，一阶均差定义为：

$$f[x_0,x_m]=\frac{f(x_m)-f(x_0)}{x_m-x_0}$$

一般地，函数 f 的 k 阶均差定义为：

$$f[x_0,x_1,\cdots,x_{k-1},x_m]=\frac{f[x_0,x_1,\cdots,x_{k-2},x_m]-f[x_0,x_1,\cdots,x_{k-1}]}{x_m-x_{k-1}}$$

利用均差的牛顿插值多项式为：

$$\begin{aligned}N(x)=&f(x_0)+f[x_0,x_1](x-x_0)+\\&f[x_0,x_1,x_2](x-x_0)(x-x_1)+\cdots+\\&f[x_0,x_1,\cdots,x_n](x-x_0)(x-x_1)\cdots(x-x_{n-1})\end{aligned}$$

系数的计算过程如表 4-2 所示。

表 4-2　均差计算表格

	一阶均差	二阶均差	三阶均差	…	n 阶均差
$f(x_0)$					
$f(x_1)$	$f[x_0,x_1]$				

续表

	一阶均差	二阶均差	三阶均差	· · ·	n 阶均差
$f(x_2)$	$f[x_0,x_2]$	$f[x_0,x_1,x_2]$			
$f(x_3)$	$f[x_0,x_3]$	$f[x_0,x_1,x_3]$	$f[x_0,x_1,x_2,x_3]$		
· · ·	· · ·	· · ·	· · ·		
$f(x_n)$	$f[x_0,x_n]$	$f[x_0,x_1,x_n]$	$f[x_0,x_1,x_2,x_n]$	· · ·	$f[x_0,x_1,\cdots,x_n]$

说明 表 4-2 中每一项均差的计算公式分析如下，设第 i 行第 j 列的均差的值为 $A(i,j)$，其中 $i=1,2,\cdots,n$， $j=1,2,\cdots,n$。

例如， $A(1,1)=f[x_0,x_1]$， $A(3,2)=f[x_0,x_1,x_3]$

则： $A(i,j)=\dfrac{A(i,j-1)-A(j-1,j-1)}{x_i-x_{j-1}}$

表 4-2 中对角线上的元素就是插值多项中对应的系数。

在 MATLAB 中编程实现的均差形式的牛顿插值法函数为：Newton

功能：求已知数据点的均差形式的牛顿插值多项式

调用格式：[f ,f0] = Newton (x,y,x0)

其中，x： 已知数据点的 x 坐标向量；

y： 已知数据点的 y 坐标向量；

x0： 插值点的 x 坐标；

f： 求得的均差形式的牛顿插值多项式

f0： x0 处的插值。

在 MATLAB 中实现利用均差的牛顿插值的代码如下所示：

```
function [f,f0] = Newton(x,y,x0)
%求已知数据点的均差形式的牛顿插值多项式
%已知数据点的 x 坐标向量：x
%已知数据点的 y 坐标向量：y
%插值点的 x 坐标：x0
%求得的均差形式的牛顿插值多项式：f
%x0 处的插值：f0
syms t;
if(length(x) == length(y))
   n = length(x);
   c(1:n) = 0.0;
else
   disp('x 和 y 的维数不相等！');
   return;
end

f = y(1);
y1 = 0;
```

```
l  = 1;
 for(i=1:n-1)
    for(j=i+1:n)
       y1(j) = (y(j)-y(i))/(x(j)-x(i));
    end
    c(i) = y1(i+1);
    l = l*(t-x(i));
    f = f + c(i)*l;
    simplify(f);
    y = y1;
end
f0 = subs(f,'t',x0);
```

例 4-3 利用均差的牛顿插值法应用实例。根据下面的数据点求出其均差形式的牛顿插值多项式，并计算当 $x=1.5$ 时 y 的值。

x	1	2	3	4	5	6
y	0	0.6931	1.0986	1.3863	1.6094	1.7918

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> x=1:6;
>> y=[0    0.6931    1.0986    1.3863    1.6094    1.7918];
>> [f,f0]=Newton(x,y,1.5)
f =6243314768165359/9007199254740992*t-6243314768165359/9007199254740992-
1295604874295013/9007199254740992*(t-1)*(t-2)+1020209651736479/360287970189
63968*(t-1)*(t-2)*(t-3)-700487017407885/1441151880758558722*(t-1)*(t-2)*(t-3)*
(t-4)+418541896262117/5764607523034234888*(t-1)*(t-2)*(t-3)*(t-4)*(t-5)
%计算 x=1.5 时的插值输出值 f0
f0 =    0.4001
```

表格中的数据点是按自然对数函数 $y=\ln x$ 给出的，而 $\ln(1.5)=0.4055$，从插值结果看出，插值的精度还是不错的，但是一旦插值点的坐标大于 6，误差就会迅速增大：

```
>> [f,f0]=Newton(x,y,10)
f = 6243314768165359/9007199254740992*t-6243314768165359/9007199254740992-
1295604874295013/9007199254740992*(t-1)*(t-2)+1020209651736479/360287970189
63968*(t-1)*(t-2)*(t-3)-700487017407885/1441151880758558722*(t-1)*(t-2)*(t-3)*
(t-4)+418541896262117/5764607523034234888*(t-1)*(t-2)*(t-3)*(t-4)*(t-5)
%计算 x=10 时的插值输出值 f0
f0 =    6.4328
>> log(10)
ans =
   2.3026
```

其实这是所有插值方法都会遇到的问题，正因为这点，所以这里介绍的插值方法都只能叫内插值，即插值点在已知点的内部。外插值有一些算法，读者可以自己去查看相关资料。

从插值多项式的形式可以看出，当增加一个插值点时，已有的插值多项式各项的形式不会发生变化，只需增加一项即可，这是均差型牛顿插值的最大优点，因此在实际应用中，如果想通过增加插值点提高插值的精度，用这种方法是最合适不过了。

4.4 等距节点插值

在实际应用中，如果节点之间是等距的话，那么利用差分可得到以下一些很方便的插值方法。

对于实际应用来说，需要对离散数据进行分析和处理，但是在数学理论中做的都是连续性的数据处理，因此在对于数学基本概念上的离散化理解就格外重要，于是引入了向前差分、向后差分、中心差分以及差商的概念。利用这些概念，我们可以把在连续数学论中的数学理论知识离散化到插值计算中，在后面的样条插值算法中，这样的概念起到了相当重要的作用。

4.4.1 利用差分的牛顿插值

差分分为前向差分、后向差分和中心差分三种，它们的记法及定义如下所示：

n 阶前向差分公式：$\Delta^n f(x_i)=\Delta^{n-1}f(x_{i+1})-\Delta^{n-1}f(x_i)$

n 阶后向差分公式：$\nabla^n f(x_i)=\nabla^{n-1}f(x_i)-\nabla^{n-1}f(x_{i-1})$

n 阶中心差分公式：$\delta^n f(x_i)=\delta^{n-1}f(x_{i+\frac{1}{2}})-\delta^{n-1}f(x_{i-\frac{1}{2}})$

其中：Δ – 前向差分；∇ – 后向差分；δ – 中心差分。

假设 $\Delta^0 f(x_i)=\nabla^0 f(x_i)=\delta^0 f(x_i)=f(x_i)$。为了方便计算，可构造如表 4-3 所示的差分表（ $f_i=f(x_i)$ ）。

表 4-3 差分计算表格

	$\Delta(\nabla,\delta)$	$\Delta^2(\nabla^2,\delta^2)$	$\Delta^3(\nabla^3,\delta^3)$	…
$f(x_0)$				
$f(x_1)$	$\Delta f_0(\nabla f_1,\delta f_{1/2})$			
$f(x_2)$	$\Delta f_1(\nabla f_2,\delta f_{3/2})$	$\Delta^2 f_0(\nabla^2 f_2,\delta^2 f_1)$		
$f(x_3)$	$\Delta f_2(\nabla f_3,\delta f_{5/2})$	$\Delta^2 f_1(\nabla^2 f_3,\delta^2 f_2)$	$\Delta^3 f_0(\nabla^3 f_3,\delta^3 f_{3/2})$	
…	…	…	…	…


表中每一项差分的计算公式分析如下，设第 i 行第 j 列的均差的值为 $A(i,j)$，其中 $i=1,2,\cdots,n$，$j=1,2,\cdots,n$。

例如，$A(1,1)=\Delta f_0(\nabla f_1,\delta f_{1/2})$，$A(3,2)=\Delta^2 f_1(\nabla^2 f_3,\delta^2 f_2)$

则：$A(i,j)=A(i,j-1)-A(i-1,j-1)$。

根据使用差分格式的不同，差分形式的牛顿差值又可细分为以下两种具体的差分形式：

1. 前向牛顿插值

前向牛顿插值多项式可表示为如下形式：

$$N(x)=N(x_0+th)$$
$$=f(x_0)+\binom{t}{1}\Delta f(x_0)+\binom{t}{2}\Delta^2 f(x_0)+\cdots+\binom{t}{n}\Delta^n f(x_0)$$

其中 h 为步长，$h=x_1-x_0$，且 t 的取值范围为 $0\leqslant t\leqslant n$。

在 MATLAB 中编程实现的前向牛顿差分插值法函数为：Newtonforward

功能：求已知数据点的前向牛顿差分插值多项式

调用格式：[f,f0]= Newtonforward (x,y,x0)

其中，x：　已知数据点的 x 坐标向量；

y：　已知数据点的 y 坐标向量；

x0：　插值点的 x 坐标；

f：　求得的前向牛顿差分插值多项式；

f0：　x0 处的插值。

在 MATLAB 中实现前向牛顿差分插值的代码如下所示：

```
function [f,f0] = Newtonforward(x,y,x0)
        %求已知数据点的前向牛顿差分插值多项式
        %已知数据点的 x 坐标向量：x
        %已知数据点的 y 坐标向量：y
        %插值点的 x 坐标：x0
        %求得的前向牛顿差分插值多项式
        %x0 处的插值：f0
syms t;
if(length(x) == length(y))
    n = length(x);
    c(1:n) = 0.0;
else
    disp('x 和 y 的维数不相等！');
    return;
end
f = y(1);
y1 = 0;
xx =linspace(x(1),x(n),(x(2)-x(1)));
if(xx ~= x)
    disp('节点之间不是等距的！');
    return;
end
for(i=1:n-1)
    for(j=1:n-i)
        y1(j) = y(j+1)-y(j);
```

```
        end
        c(i) = y1(1);
        l = t;
        for(k=1:i-1)
            l = l*(t-k);
        end;

        f = f + c(i)*l/factorial(i);
        simplify(f);
        y = y1;
    end
    f0 = subs(f,'t',(x0-x(1))/(x(2)-x(1)));
```

2. 后向牛顿插值

后向牛顿插值多项式可表示为如下形式：

$$N(x) = N(x_0 + th)$$
$$= f(x_n) + \binom{t}{1}\nabla f(x_n) + \binom{t+1}{2}\nabla^2 f(x_n) + \cdots + \binom{t+n-1}{n}\nabla^n f(x_n)$$

h 仍然是步长，$h=x_1-x_0$，但 t 的取值范围为 $-n \leqslant t \leqslant 0$。

在 MATLAB 中编程实现的后向牛顿差分插值法函数为：Newtonback

功能：求已知数据点的后向牛顿差分插值多项式

调用格式：[f,f0] = Newtonback (x,y,x0)

其中，x：　已知数据点的 x 坐标向量；

y：　已知数据点的 y 坐标向量；

x0：　插值点的 x 坐标；

f：　求得的后向牛顿差分插值多项式；

f0：　x0 处的插值。

在 MATLAB 中实现后向牛顿差分插值的代码如下所示：

```
function [f,f0] = Newtonback(x,y,x0)
%求已知数据点的后向牛顿差分插值多项式
%已知数据点的 x 坐标向量：x
%已知数据点的 y 坐标向量：y
%插值点的 x 坐标：x0
%求得的后向牛顿差分插值多项式
%x0 处的插值：f0
syms t;
if(length(x) == length(y))
    n = length(x);
    c(1:n) = 0.0;
else
    disp('x 和 y 的维数不相等！');
```

```
        return;
    end
    f = y(n);
    y1 = 0;
    xx =linspace(x(1),x(n),(x(2)-x(1)));
    if(xx ~= x)
        disp('节点之间不是等距的! ');
        return;
    end
    for(i=1:n-1)
        for(j=i+1:n)
            y1(j) = y(j)-y(j-1);
        end
        c(i) = y1(n);
        l = t;
        for(k=1:i-1)
            l = l*(t+k);
        end;

        f = f + c(i)*l/factorial(i);
        simplify(f);
        y = y1;
    end
    f0 = subs(f,'t',(x(n)-x0)/(x(2)-x(1)));
```

例 4-4 利用差分的牛顿插值法应用实例。根据下面的数据点求出其差分形式的牛顿插值多项式，并计算当 $x=1.55$ 时 y 的值。

x	1	1.2	1.4	1.6	1.8
y	0.8415	0.9320	0.9854	0.9996	0.9738

解：在 MATLAB 命令窗口中输入以下命令：

```
>> x=1:0.2:1.8;
>> y=[0.8415  0.9320  0.9854  0.9996  0.9738];
>> [f,f0]=Newtonforward(x,y,1.55)
f = 1683/2000+181/2000*t-371/20000*t*(t-1)-4728779608739/13510798882111488*
t*(t-1)*(t-2)+11709359031163/216172782113783808*t*(t-1)*(t-2)*(t-3)
%计算 x=1.55 时的插值输出值 f0
f0 =    0.9998
>> [f,f0]=Newtonback(x,y,1.55)
f =4869/5000-129/5000*t-1/50*t*(t+1)-7205759403793/54043195528445952*t*
(t+1)*(t+2)+11709359031163/216172782113783808*t*(t+1)*(t+2)*(t+3)
%计算 x=1.55 时的插值输出值 f0
f0 =    0.9998
```

表格中的数据点是按 $y=\sin x$ 给出的，而 sin(1.55)=0.9998，从插值函数在 x=1.55 的值看出，牛顿差分插值函数是很准确的。

顾名思义，在使用牛顿差分插值公式时，如果插值点离 x_0 比较近，则适合用牛顿前插公式，否则建议用牛顿后插公式；从例子的结果也可以看出，对于同一个插值点，不管用前插公式还是后插公式，得出的结果都是一样的：

```
>> [f,f0]=Newtonback(x,y,1.22)  %计算插值点为 1.22 时的结果
f =4869/5000-129/5000*t-1/50*t*(t+1)-7205759403793/54043195528445952*t*
(t+1)*(t+2)+11709359031163/216172782113783808*t*(t+1)*(t+2)*(t+3)
f0 =    0.9391
>> [f,f0]=Newtonforward(x,y,1.22) %计算插值点为 1.22 时的结果
f =1683/2000+181/2000*t-371/20000*t*(t-1)-4728779608739/13510798882111488*t*
(t-1)*(t-2)+11709359031163/216172782113783808*t*(t-1)*(t-2)*(t-3)
f0 =    0.9391
```

结果都是一样的，读者还可以试试其他的值。

4.4.2　高斯插值

高斯插值是从中间的节点开始的，它也有前向和后向之分。

1．前向高斯插值

对于节点个数为偶数的情况（ $n=2m$ ），设节点编号为 $x_i,x_{i+1},x_{i-1},\cdots,x_{i+m},x_{i-m}$ ，前向高斯插值公式为：

$$G(x)=f(x_i)+\binom{t}{1}\delta f(x_{i+\frac{1}{2}})+\binom{t}{2}\delta^2 f(x_i)+\binom{t+1}{3}\delta^3 f(x_{i+\frac{1}{2}})+\cdots$$
$$+\binom{t+m-1}{2m-1}\delta^{2m-1} f(x_{i+\frac{1}{2}})+\binom{t+m-1}{2m}\delta^{2m} f(x_i)$$

对于节点个数为奇数的情况（ $n=2m+1$ ），设节点编号为 $x_i,x_{i+1},x_{i-1},\cdots,x_{i+m},x_{i-m},x_{i+m+1}$ ，高斯插值公式为：

$$G(x)=f(x_i)+\binom{t}{1}\delta f(x_{i+\frac{1}{2}})+\binom{t}{2}\delta^2 f(x_i)+\binom{t+1}{3}\delta^3 f(x_{i+\frac{1}{2}})+\cdots$$
$$+\binom{t+m-1}{2m-1}\delta^{2m-1} f(x_{i+\frac{1}{2}})+\binom{t+m-1}{2m}\delta^{2m} f(x_i)+\binom{t+m}{2m+1}\delta^{2m+1} f(x_{i+\frac{1}{2}})$$

2．后向高斯插值

对于节点个数为偶数的情况（ $n=2m$ ），设节点编号为 $x_i,x_{i-1},x_{i+1},\cdots,x_{i-m},x_{i+m}$ ，注意节点编号顺序，后向高斯插值公式为：

$$G(x)=f(x_i)+\binom{t}{1}\delta f(x_{i-\frac{1}{2}})+\binom{t+1}{2}\delta^2 f(x_i)+\binom{t+1}{3}\delta^3 f(x_{i-\frac{1}{2}})+\cdots$$
$$+\binom{t+m-1}{2m-1}\delta^{2m-1} f(x_{i-\frac{1}{2}})+\binom{t+m}{2m}\delta^{2m} f(x_i)$$

对于节点个数为奇数的情况（$n=2m+1$），设节点编号为 $x_i, x_{i-1}, x_{i+1}, \cdots, x_{i-m}, x_{i+m}, x_{i-m-1}$，高斯插值公式为：

$$G(x)=f(x_i)+\binom{t}{1}\delta f(x_{i-\frac{1}{2}})+\binom{t+1}{2}\delta^2 f(x_i)+\binom{t+1}{3}\delta^3 f(x_{i-\frac{1}{2}})+\cdots$$
$$+\binom{t+m-1}{2m-1}\delta^{2m-1} f(x_{i-\frac{1}{2}})+\binom{t+m}{2m}\delta^{2m} f(x_i)+\binom{t+m}{2m+1}\delta^{2m+1} f(x_{i-\frac{1}{2}})$$

实际应用得比较多的是由高斯公式衍生出来的斯特林公式和贝塞尔公式：

（1）用于奇数个节点插值的斯特林公式

斯特林公式是对前向高斯插值和后向高斯插值取平均得到的：

$$S(x)=f(x_i)+\frac{1}{2}\binom{t}{1}(\delta f(x_{i+\frac{1}{2}})+\delta f(x_{i-\frac{1}{2}}))+\frac{1}{2}[\binom{t}{2}+\binom{t+1}{2}]\delta^2 f(x_i)+$$
$$\frac{1}{2}\binom{t+1}{3}(\delta^3 f(x_{i+\frac{1}{2}})+\delta^3 f(x_{i-\frac{1}{2}}))+\cdots+$$
$$\frac{1}{2}[\binom{t+m-1}{2m-1}+\binom{t+m}{2m-1}]\delta^{2m-1} f(x_{i+\frac{1}{2}})+\binom{t+m-1}{2m}\delta^{2m} f(x_i)$$

这个公式只适合于奇数个节点插值。

（2）用于偶数个节点插值的贝塞尔公式

贝塞尔公式是高斯公式的另一种平均：

$$B(x)=\frac{1}{2}(f(x_i)+f(x_{i+1}))+\frac{1}{2}[\binom{t}{1}+\binom{t-1}{1}]\delta f(x_{i+\frac{1}{2}})+$$
$$\frac{1}{2}\binom{t}{2}(\delta^2 f(x_i)+\delta^2 f(x_{i+1}))+$$
$$\frac{1}{2}[\binom{t+1}{3}+\binom{t}{3}]\delta^3 f(x_{i+\frac{1}{2}})+\cdots+$$
$$\frac{1}{2}[\binom{t+m-1}{2m+1}+\binom{t+m-1}{2m+1}]\delta^{2m+1} f(x_{i+\frac{1}{2}})$$

这个公式只适合于偶数个节点插值。

在 MATLAB 中编程实现的高斯插值法函数为：Gauss

功能：求已知数据点的高斯插值多项式

调用格式：[f,f0]= Gauss (x,y,x0)

其中，x：　已知数据点的 x 坐标向量；

y：　已知数据点的 y 坐标向量；

x0：　插值点的 x 坐标；

f:	求得的高斯插值多项式;
f0:	x0 处的插值。

在 MATLAB 中实现高斯插值的代码如下所示:

```
function [f,f0] = Gauss(x,y,x0)
%求已知数据点的高斯插值多项式
%已知数据点的 x 坐标向量: x
%已知数据点的 y 坐标向量: y
%插值点的 x 坐标: x0
%求得的高斯插值多项式: f
%x0 处的插值: f0
if(length(x) == length(y))
   n = length(x);
   end
else
   disp('x 和 y 的维数不相等! ');
   return;
end
xx =linspace(x(1),x(n),(x(2)-x(1)));
if(xx ~= x)
   disp('节点之间不是等距的! ');
   return;
end
if( mod(n,2) ==1)                               %n 为奇数时采用斯特林插值
      [f,f0] = GStirling(x,y,n,x0);
else                                            %n 为偶数时采用贝塞尔插值
      [f,f0] = GBessel(x,y,n,x0);
end
function [f,f0] = GStirling(x,y,n,x0)           %斯特林插值
syms t;
nn = (n+1)/2;
f = y(nn);
for(i=1:n-1)
   for(j=i+1:n)
   . y1(j) = y(j)-y(j-1);
   end
   if(mod(i,2)==1)
      c(i) = (y1((i+n)/2)+y1((i+n+2)/2))/2;
   else
      c(i) = y1((i+n+1)/2)/2;
   end
   if(mod(i,2)==1)
      l = t+(i-1)/2;
      for(k=1:i-1)
         l = l*(t+(i-1)/2-k);
      end
   else
```

```
            l_1 = t+i/2-1;
            l_2 = t+i/2;
            for(k=1:i-1)
                l_1 = l_1*(t+i/2-1-k);
                l_2 = l_2*(t+i/2-k);
            end
            l = l_1 + l_2;
        end
        l = l/factorial(i);
        f = f + c(i)*l;
        simplify(f);
        y = y1;
    end
    f0 = subs(f,'t',(x0-x(nn))/(x(2)-x(1)));
    function [f,f0] = GBessel(x,y,n,x0)                    %贝塞尔插值
    syms t;
    nn = n/2;
    f = (y(nn)+y(nn+1))/2;
    for(i=1:n-1)
        for(j=i+1:n)
            y1(j) = y(j)-y(j-1);
        end
        if(mod(i,2)==1)
            c(i) = y1((i+n+1)/2)/2;
        else
            c(i) = (y1((i+n)/2)+y1((i+n+2)/2))/2;
        end
        if(mod(i,2)==0)
            l = t+i/2-1;
            for(k=1:i-1)
                l = l*(t+i/2-1-k);
            end
        else
            l_1 = t+(i-1)/2;
            l_2 = t+(i-1)/2-1;
            for(k=1:i-1)
                l_1 = l_1*(t+(i-1)/2-k);
                l_2 = l_2*(t+(i-1)/2-1-k);
            end
            l = l_1 + l_2;
        end
        l = l/factorial(i);
        f = f + c(i)*l;
        simplify(f);
        y = y1;
    end
    f0 = subs(f,'t',(x0-x(nn))/(x(2)-x(1)));
```

例 4-5 高斯插值法应用实例 1。根据下面的数据点求出其高斯插值多项式，并计算当 $x=1.5$ 时 y 的值。

x	1	1.2	1.4	1.6	1.8
y	0	0.2630	0.4854	0.6781	0.8480

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> x=1:0.2:1.8;
>> y=[0  0.2630  0.4854  0.6781  0.8480];
>> [f,f0]=Gauss(x,y,1.5)
f =4372336155919085/9007199254740992+233645340887123/1125899906842624*t-66985062312357/9007199254740992*t*(t-1)-66985062312357/9007199254740992*t*(t+1)+40356549089119/27021597764222976*t*(t+1)*(t-1)-17417105048023/216172782113783808*t*(t+1)*(t-1)*(t-2)-17417105048023/216172782113783808*t*(t+1)*(t+2)*(t-1)
%计算 x=1.5 时的插值输出值 f0
f0 =    0.5849
>> log2(1.5)
ans =    0.5850
```

由于 x 和 y 是奇数维的，高斯函数将调用斯特林公式插值。表格中的数据点是按 $y=\log_2 x$ 给出的，而 $\log_2 1.5=0.5850$。

例 4-6 高斯插值法应用实例 2。根据下面的数据点求出其高斯插值多项式，并计算当 $x=1.5$ 时 y 的值。

x	1	1.2	1.4	1.6	1.8	2
y	0	0.2630	0.4854	0.6781	0.8480	1.000

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> x=1:0.2:2;
>> y=[0  0.2630  0.4854  0.6781  0.8480 1.000];
>> [f,f0]=Gauss(x,y,1.5)
f =4372336155919085/9007199254740992+867596301236135/4503599627370496*t-472584505368641/36028797018963968*t*(t-1)+63295993130215/108086391056891904*t*(t+1)*(t-1)+63295993130215/108086391056891904*t*(t-1)*(t-2)-18304047821175/144115188075855872*t*(t+1)*(t-1)*(t-2)+14756276728567/2161727821137838080*t*(t+1)*(t+2)*(t-1)*(t-2)+14756276728567/2161727821137838080*t*(t+1)*(t-1)*(t-2)*(t-3)
%计算 x=1.5 时的插值输出值 f0
f0 =    0.5850
```

由于 x 和 y 是偶数维的，高斯函数将调用贝塞尔公式插值。表格中的数据点是按 $y=\log_2 x$ 给出的，而 $\log_2 1.5=0.5850$。

4.5 埃尔米特插值

埃尔米特插值法满足在节点上等于给定函数值，而且在节点上的导数值也等于给定的导数值，对于有高阶导数的情况，埃尔米特插值多项比较复杂，在实际应用中，常常遇到的是函数值与一阶导数值给定的情况，在这种情况下，n 个节点 $x_1,x_2\cdots,x_n$ 的埃尔米特插值多项式 $H(x)$ 的表达式如下所示：

$$H(x)=\sum_{i=1}^{n}h_i[(x_i-x)(2a_iy_i-y_i^{'})+y_i]$$

其中 $y_i=y(x_i),y_i^{'}=y^{'}(x_i)$

$$h_i=\prod_{\substack{j=1\\j\neq i}}^{n}(\frac{x-x_j}{x_i-x_j})^2,a_i=\sum_{\substack{j=1\\j\neq i}}^{n}\frac{1}{x_i-x_j}$$

在 MATLAB 中编程实现的埃尔米特插值法函数为：Hermite

功能：求已知数据点的埃尔米特插值多项式

调用格式：[f,f0] = Hermite (x,y,y_1,x0)

其中，x：　已知数据点的 x 坐标向量；

y：　已知数据点的 y 坐标向量；

y_1：　已知数据点的导数向量；

x0：　插值点的 x 坐标；

f：　求得的埃尔米特插值多项式；

f0：　x0 处的插值。

在 MATLAB 中实现埃尔米特插值的代码如下所示：

```
function [f,f0]= Hermite(x,y,y_1,x0)
%求已知数据点的埃尔米特插值多项式
%已知数据点的 x 坐标向量：x
%已知数据点的 y 坐标向量：y
%已知数据点的导数向量：y_1
%插值点的 x 坐标：x0
%求得的埃尔米特插值多项式：f
% x0 处的插值：f0
syms t;
f = 0.0;
if(length(x) == length(y))
    if(length(y) == length(y_1))
        n = length(x);
    else
        disp('y 和 y 的导数的维数不相等！');
        return;
```

```
        end
    else
        disp('x和y的维数不相等！');
        return;
    end
    for i=1:n
        h = 1.0;
        a = 0.0;
        for j=1:n
            if( j ~= i)
                h = h*(t-x(j))^2/((x(i)-x(j))^2);
                a = a + 1/(x(i)-x(j));
            end
        end
        f = f + h*((x(i)-t)*(2*a*y(i)-y_1(i))+y(i));
    end
    f0 = subs(f,'t',x0);
```

例 4-7　埃尔米特插值法应用实例。根据下面的数据点求出其埃尔米特插值多项式，并计算当 $x=1.44$ 时 y 的值。

x	1	1.2	1.4	1.6	1.8
y	1	1.0954	1.1832	1.2649	1.3416
y'	0.5000	0.4564	0.4226	0.3953	0.3727

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> x=1:0.2:1.8;
>> y=[1  1.0954  1.1832  1.2649 1.3416];
>> y_1=[0.5  0.4564 0.4226 0.3953 0.3727];
>> [f,f0]=Hermite(x,y,y_1,1.44)
f =     674.168*(t-1.20000)^2*(t-1.40000)^2*(t-1.60000)^2*(t-1.80000)^2*
(-20.3333+21.3333*t)+10850.7*(t-1.)^2*(t-1.40000)^2*(t-1.60000)^2*(t-1.80000
)^2*(-10.4066+9.58511*t)+24414.1*(t-1.)^2*(t-1.20000)^2*(t-1.60000)^2*(t-1.8
0000)^2*(.591580+.422600*t)+10850.7*(t-1.)^2*(t-1.20000)^2*(t-1.40000)^2*(t-
1.80000)^2*(17.4979-10.1456*t)+678.168*(t-1.)^2*(t-1.20000)^2*(t-1.40000)^2*
(t-1.60000)^2*(50.9823-27.5781*t)
%计算 x=1.44 时的插值输出值 f0
f0 =    1.2000
```

表格中的数据点是按 $y=\sqrt{x}$ 给出的，而 $\sqrt{1.44}=1.20$，从插值函数在 x=1.44 的值看出，插值结果很准确。

4.6 分段三次埃尔米特插值

给定插值节点 x_i、节点函数值 y_i 及对应的导数值 y_i' $(i=0,1,2,\cdots,N)$，则满足下面条件：

$$p(x_i)=y_i, p'(x_i)=y_i'$$

的分段埃尔米特插值函数 $p(x)$ 的表达式如下所示：

$$p_i(x)=y_i(1+2\frac{x-x_i}{h_i})(\frac{x-x_{i+1}}{h_i})^2+y_{i+1}(1+2\frac{x-x_{i+1}}{-h_i})(\frac{x-x_i}{h_i})^2+$$

$$y_i'(x-x_i)(\frac{x-x_{i+1}}{h_i})^2+y_{i+1}'(x-x_{i+1})(\frac{x-x_i}{h_i})^2$$

$$h_i=x_{i+1}-x_i,(i=0,1,\cdots,N-1),x\in[x_i,x_{i+1}]$$

在 MATLAB 中编程实现的分段三次埃尔米特插值法函数为：SubHermite

功能：求已知数据点的分段三次埃尔米特插值多项式及其插值点处的值

调用格式：[f,f0] = SubHermite (x,y,y_1,x0)

其中，x：　已知数据点的 x 坐标向量；

y：　已知数据点的 y 坐标向量；

y_1：　已知数据点的导数向量；

x0：　插值点的 x 坐标；

f：　求得的分段三次埃尔米特插值多项式；

f0：　x0 处的插值。

在 MATLAB 中实现分段三次埃尔米特插值的代码如下所示：

```
function [f,f0] = SubHermite(x,y,y_1,x0)
%求已知数据点的分段三次埃尔米特插值多项式及其插值点处的值
%已知数据点的 x 坐标向量：x
%已知数据点的 y 坐标向量：y
%已知数据点的导数向量：y_1
%插值点的 x 坐标：x0
%求得的分段三次埃尔米特插值多项式：f
%求得的 x0 处的插值：f0
syms t;
f = 0.0;
f0 = 0.0;
if(length(x) == length(y))
    if(length(y) == length(y_1))
        n = length(x);
    else
        disp('y 和 y 的导数的维数不相等！');
        return;
    end
else
    disp('x 和 y 的维数不相等！');
    return;
end                    %维数检查
for i=1:n
    if(x(i)<=x0)&& (x(i+1)>=x0)
```

```
        index = i;
        break;
    end
end                          %找到 x0 所在区间
h = x(index+1) - x(index);   %x0 所在区间长度
f1 = y(index)*(1+2*(t-x(index))/h)*(t-x(index+1))^2/h/h + ...
        y(index+1)*(1-2*(t-x(index+1))/h)*(t-x(index))^2/h/h;
fr = y_1(index)*(t-x(index))*(t-x(index+1))^2/h/h + ...
        y_1(index+1)*(t-x(index+1))*(t-x(index))^2/h/h;
f = fl + fr;                 %x0 所在区间的插值函数
f0 = subs(f,'t',x0);         %x0 处的插值
```

例 4-8 分段埃尔米特插值法应用实例。根据下面的数据点求出其分段埃尔米特插值多项式，并计算当 $x=1.69$ 时 y 的值。

x	1	1.2	1.4	1.6	1.8
y	1	1.0954	1.1832	1.2649	1.3416
y'	0.5000	0.4564	0.4226	0.3953	0.3727

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> x=1:0.2:1.8;
>> y=[1  1.0954  1.1832  1.2649 1.3416];
>> y_1=[0.5  0.4564 0.4226 0.3953 0.3727]
>> [f,f0]=SubHermite(x,y,y_1,1.69)
f = 25*(-37947/2000+12649/1000*t)*(t-9/5)^2+25*(31863/1250-1677/125*t)*
(t-8/5)^2+25*(3953/10000*t-3953/6250)*(t-9/5)^2+25*(3727/10000*t-33543/50000
)*(t-8/5)^2
%计算 x=1.69 时的插值输出值 f0
f0 =    1.3000;
```

表格中的数据点是按 $y=\sqrt{x}$ 给出的，而 $\sqrt{1.69}=1.30$，和上一例题相比较，插值结果的精度没下降，但是插值多项式却短了许多。

4.7 样条插值

应用广泛的样条插值有二次样条插值函数、三次样条插值和 B 样条插值。区别在于插值的多项式次数不同，以及需要的待定参数与插值条件不同。它们的插值条件都可知，它们的插值方法都是唯一的，而且是可解的。

4.7.1 二次样条插值

给定插值节点 x_i 及对应的导数值 y_i' $(i=0,1,2,\cdots,N)$，端点 x_0 和端点的函数值 y_0，则满足下面条件：

$$p(x_0)=y_0, p'(x_i)=y_i'$$

的分段函数 $p(x)$ 的表达式如下所示：

$$p_i(x)=\frac{(x-x_i)^2}{2h_i}y'_{i+1}-\frac{(x_{i+1}-x)^2}{2h_i}y'_i+d_i,\ (i=0,1,\cdots,N-1),x\in[x_i,x_{i+1}]$$

其中，

$$\begin{cases}h_i=x_{i+1}-x_i,(i=0,1,2,\cdots N-1)\\ d_0=\dfrac{h_0}{2}y'_0+y_0\\ d_i=d_{i-1}+y'_i(h_i+h_{i-1})/2,(i=1,2,\cdots N-1)\end{cases}$$

在 MATLAB 中编程实现的二次样条插值法函数为：SecSample

功能：求已知数据点的二次样条插值多项式及其插值点处的值

调用格式：[f,f0,fd0] = SecSample (x,y,y_1,x0)

其中，x：　已知数据点的 x 坐标向量；

y：　已知数据点的 y 坐标向量；

y_1：　已知数据点的导数向量；

x0：　插值点的 x 坐标；

f：　求得的二次样条插值多项式；

f0：　求得的 x0 处的插值；

fd0：　求得的 x0 处的导数的插值。

在 MATLAB 中实现二次样条插值法的代码如下所示：

```
function [f,f0,fd0] = SecSample (x,y,y_1,x0)
%求已知数据点的二次样条插值多项式及其插值点处的值
%已知数据点的 x 坐标向量: x
%已知数据点的 y 坐标向量: y
%已知数据点的导数向量: y_1
%插值点的 x 坐标: x0
%求得的二次样条插值多项式: f
%求得的 x0 处的插值: f0
%求得的 x0 处的导数的插值:  fd0
syms t;
f = 0.0;
f0 = 0.0;
if(length(x) == length(y))
    if(length(y) == length(y_1))
        n = length(x);
    else
        disp('y 和 y 的导数的维数不相等! ');
        return;
    end
else
    disp('x 和 y 的维数不相等! ');
    return;
```

```
end                    %维数检查
for i=1:n
    if(x(i)<=x0)&& (x(i+1)>=x0)
        index = i;
        break;
    end
end                          %找到 x0 所在区间
d = y_1(1)*(x(2)-x(1))/2+y(1);
for i=2:n-1
    d = d + y_1(i)*(x(i+1)-x(i-1))/2;
end
h = x(index+1) - x(index);   %x0 所在区间长度
f = y_1(index+1)*(t-x(index))^2/2/h + ...
       y_1(index)*(t-x(index+1))^2/2/h + d;     %x0 所在区间的插值函数
fd = (t-x(index))*y_1(index+1)/h + y_1(index)*(x(index+1)-t)/h;
                                         %x0 所在区间的插值函数的导数
f0 = subs(f,'t',x0);          %x0 处的插值
fd0 = subs(fd,'t',x0);       % x0 处的导数插值
```

例 4-9 二次样条插值应用实例。根据下面的数据点求出二次样条插值多项式，并计算当 $x=4.3$ 时 y 的值。

x	1	2	3	4	5	6	7	8
y	0.8415	0.9093	0.1411	−0.7568	−0.9589	−0.2794	0.6570	0.9894
y'	0.5403	−0.4161	−0.9900	−0.6536	0.2837	0.9602	0.7539	−0.1455

解：在 MATLAB 命令窗口中输入以下命令：

```
>> x=1:8;
>> y=[ 0.8415    0.9093    0.1411   -0.7568   -0.9589   -0.2794    0.6570
    0.9894];
>> y_1=[ 0.5403   -0.4161   -0.9900   -0.6536    0.2837    0.9602    0.7539
   -0.1455]
>> [f,f0,fd0]=SecSample(x,y,y_1,4.3)
f =5110003651005145/36028797018963968*(t-4)^2-5887498334708929/18014398
509481984*(t-5)^2+1181715167704739/1125899906842624
%计算 x=4.3 时的插值输出值 f0
f0 =    0.9022
fd0 =   -0.3725
```

表格中的数据点是按 $y=\sin x$ 给出的，而 $\sin 4.3=-0.9162$，$(\sin x)'\big|_{x=4.3}=\cos 4.3=-0.4008$，从二次样条满足的插值条件就可解释上面得出的结果为什么是这样，在实际应用中，二次样条基本上没什么用。样条里有用的是三次样条，在图形学、曲线拟合等方面有极其重要的作用。

4.7.2 三次样条插值

给定插值节点 x_i 及对应的函数值 y_i $(i=0,1,2,\cdots,N)$、左右两端点的函数值及各阶导

数值，则分别满足下面三个条件：

（1）第一类边界条件：$p(x_i)=y_i,(i=0,1,2,\cdots,N)$

$p'(x_0)=y'_0, p'(x_N)=y'_N$

（2）第二类边界条件：$p(x_i)=y_i,(i=0,1,2,\cdots,N)$

$p''(x_0)=y''_0, p''(x_N)=y''_N$

（3）第三类边界条件：

$p(x_i)=y_i,(i=1,2,\cdots,N-1)$

$p(x_0)=p(x_N)$

$p'(x_0)=p'(x_N)$

$p''(x_0)=p''(x_N)$

的分段函数 $p(x)$ 的表达式如下：

$$
\begin{aligned}
p_i(x)=&\frac{y_i}{h_i^3}[2(x-x_i)+h_i](x_{i+1}-x)^2\\
&+\frac{y_{i+1}}{h_i^3}[2(x_{i+1}-x)+h_i](x-x_i)^2\\
&+\frac{m_i}{h_i^2}(x-x_i)(x_{i+1}-x)^2-\frac{m_{i+1}}{h_i^2}(x_{i+1}-x)(x-x_i)^2
\end{aligned}
$$

$$h_i=x_{i+1}-x_i(i=0,1,\cdots,N-1),x\in[x_i,x_{i+1}]$$

（1）对于第一类边界条件，有

$$
\begin{bmatrix}
2 & \mu_0 & & & \\
\lambda_1 & 2 & \mu_1 & & \\
 & \lambda_2 & \ddots & \ddots & \\
 & & \ddots & 2 & \mu_{N-1} \\
 & & & \lambda_N & 2
\end{bmatrix}
\begin{bmatrix} m_0 \\ m_1 \\ \vdots \\ m_{N-1} \\ m_N \end{bmatrix}
=
\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{N-1} \\ c_N \end{bmatrix}
$$

其中，

$$
\begin{cases}
\mu_0=\lambda_N=0, c_0=2y'_0, c_N=2y'_N \\
\lambda_i=\dfrac{h_i}{h_i+h_{i-1}}, \mu_i=\dfrac{h_{i-1}}{h_i+h_{i-1}} \\
c_i=\dfrac{3\lambda_i(y_i-y_{i-1})}{h_{i-1}}+\dfrac{3\mu_i(y_{i+1}-y_i)}{h_i}
\end{cases}
$$

$$(i=1,2,\cdots,N-1)$$

在 MATLAB 中编程实现的第一类三次样条插值法函数为：ThrSample1

功能：求已知数据点的第一类三次样条插值多项式及其插值点处的值

调用格式：[f,f0] = ThrSample1 (x,y,y_1,y_N,x0)

其中，x：　已知数据点的 x 坐标向量；

y：　已知数据点的 y 坐标向量；

y_1:	左端点的一阶导数;
y_N:	右端点的一阶导数;
x0:	插值点的 x 坐标;
f:	求得的三次样条插值多项式;
f0:	求得的 x0 处的插值。

在 MATLAB 中实现第一类三次样条插值的代码如下所示:

```
function [f,f0] = ThrSample1 (x,y,y_1, y_N,x0)
%求已知数据点的第一类三次样条插值多项式及其插值点处的值
%已知数据点的 x 坐标向量: x
%已知数据点的 y 坐标向量: y
%左端点的一阶导数;: y_1
%右端点的一阶导数;: y_N
%插值点的 x 坐标: x0
%求得的三次样条插值多项式: f
%求得的 x0 处的插值: f0
syms t;
f = 0.0;
f0 = 0.0;
if(length(x) == length(y))
    n = length(x);
else
    disp('x 和 y 的维数不相等! ');
    return;
end                     %维数检查
for i=1:n
    if(x(i)<=x0)&& (x(i+1)>=x0)
        index = i;
        break;
    end
end                             %找到 x0 所在区间
for i=1:n
    if(x(i)<=x0)&& (x(i+1)>=x0)
        index = i;
        break;
    end
end                             %找到 x0 所在区间
A = diag(2*ones(1,n));       %求解 m 的系数矩阵
u = zeros(n-2,1);
lamda = zeros(n-1,1);
c = zeros(n,1);
for i=2:n-1
    u(i-1) = (x(i)-x(i-1))/(x(i+1)-x(i-1));
    lamda(i) = (x(i+1)-x(i))/(x(i+1)-x(i-1));
    c(i) = 3*lamda(i)*(y(i)-y(i-1))/(x(i)-x(i-1))+ ...
```

```
        3*u(i-1)*(y(i+1)-y(i))/(x(i+1)-x(i));
    A(i, i+1) = u(i-1);
    A(i, i-1) = lamda(i);    %形成系数矩阵及向量 c
end
c(1) = 2*y_1;
c(n) = 2*y_N;
m = followup(A,c);            %用追赶法求解方程组
h = x(index+1) - x(index);   %x0 所在区间长度
f = y(index)*(2*(t-x(index))+h)*(t-x(index+1))^2/h/h/h + ...
        y(index+1)*(2*(x(index+1)-t)+h)*(t-x(index))^2/h/h/h + ...
        m(index)*(t-x(index))*(x(index+1)-t)^2/h/h - ...
        m(index+1)*(x(index+1)-t)*(t-x(index))^2/h/h;
                              %x0 所在区间的插值函数
f0 = subs(f,'t',x0);         %x0 处的插值
```

例 4-10 第一类三次样条插值应用实例。根据下面的数据点求出其第一类三次样条插值多项式，并计算当 $x=4.3$ 时 y 的值。

x	1	2	3	4	5	6	7	8
y	0.8415	0.9093	0.1411	−0.7568	−0.9589	−0.2794	0.6570	0.9894
y'	0.5403	−0.4161	−0.9900	−0.6536	0.2837	0.9602	0.7539	−0.1455

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> x=1:8;
>> y=[ 0.8415   0.9093   0.1411   -0.7568   -0.9589   -0.2794   0.6570
   0.9894];
>> [f,f0]=ThrSample1(x,y,0.5403,-0.1455,4.3)
f =(-3408335435861847/2251799813685248*t+2385834805103292 9/4503599627370496)*
(t-5)^2+(-9500944213308753 7/9007199254740992+8637222012098867/4503599627370
496*t)*(t-4)^2+(-5851300015778661/9007199254740992*t+5851300015778661/22517
99813685248)*(5-t)^2-(25393349669186305/18014398509481984-5078669933837261/
18014398509481984*t)*(t-4)^2
%计算 x=4.3 时的插值输出值 f0
f0 =   -0.9137
```

表格中的数据点是按 $y=\sin x$ 给出的，而 $\sin 4.3=-0.9162$。

（2）对于第二类边界条件，m_i 满足的方程组与（1）是一样的，不同的是

$$\begin{cases} \mu_0=\lambda_N=1 \\ c_0=\dfrac{3(y_1-y_0)}{h_0}-\dfrac{1}{2}h_0 y_0'' \\ c_N=\dfrac{3(y_N-y_{N-1})}{h_{N-1}}+\dfrac{1}{2}h_{N-1}y_N'' \end{cases} \quad (i=1,2,\cdots,N-1)$$

在 MATLAB 中编程实现的第二类三次样条插值法函数为：ThrSample2

功能：求已知数据点的第二类三次样条插值多项式及其插值点处的值

调用格式：[f,f0] = ThrSample2 (x,y,y2_1,y2_N,x0)

其中，x：　已知数据点的 x 坐标向量；

y：　已知数据点的 y 坐标向量；

y2_1：　左端点的二阶导数；

y2_N：　右端点的二阶导数；

x0：　插值点的 x 坐标；

f：　求得的三次样条插值多项式；

f0：　求得的 x0 处的插值。

在 MATLAB 中实现第二类三次样条插值法的代码如下所示：

```
function [f,f0] = ThrSample2 (x,y,y2_1, y2_N,x0)
%求已知数据点的第二类三次样条插值多项式及其插值点处的值
%已知数据点的 x 坐标向量: x
%已知数据点的 y 坐标向量: y
%左端点的二阶导数;: y2_1
%右端点的二阶导数;: y2_N
%插值点的 x 坐标: x0
%求得的插值多项式: f
%求得的 x0 处的插值: f0
syms t;
f = 0.0;
f0 = 0.0;
if(length(x) == length(y))
   n = length(x);
else
   disp('x 和 y 的维数不相等! ');
   return;
end                  %维数检查
for i=1:n
   if(x(i)<=x0)&& (x(i+1)>=x0)
      index = i;
      break;
   end
end                          %找到 x0 所在区间
for i=1:n
   if(x(i)<=x0)&& (x(i+1)>=x0)
      index = i;
      break;
   end
end                          %找到 x0 所在区间
A = diag(2*ones(1,n));         %求解 m 的系数矩阵
A(1,2) = 1;
A(n,n-1) = 1;
u = zeros(n-2,1);
```

```
lamda = zeros(n-1,1);
c = zeros(n,1);
for i=2:n-1
    u(i-1) = (x(i)-x(i-1))/(x(i+1)-x(i-1));
    lamda(i) = (x(i+1)-x(i))/(x(i+1)-x(i-1));
    c(i) = 3*lamda(i)*(y(i)-y(i-1))/(x(i)-x(i-1))+ ...
        3*u(i-1)*(y(i+1)-y(i))/(x(i+1)-x(i));
    A(i, i+1) = u(i-1);
    A(i, i-1) = lamda(i);     %形成系数矩阵及向量 c
end
c(1) = 3*(y(2)-y(1))/(x(2)-x(1))-(x(2)-x(1))*y2_1/2;
c(n) = 3*(y(n)-y(n-1))/(x(n)-x(n-1))-(x(n)-x(n-1))*y2_N/2;
m = followup(A,c);            %用追赶法求解方程组
h = x(index+1) - x(index);   %x0 所在区间长度
f = y(index)*(2*(t-x(index))+h)*(t-x(index+1))^2/h/h/h + ...
        y(index+1)*(2*(x(index+1)-t)+h)*(t-x(index))^2/h/h/h + ...
        m(index)*(t-x(index))*(x(index+1)-t)^2/h/h - ...
        m(index+1)*(x(index+1)-t)*(t-x(index))^2/h/h;
                                     %x0 所在区间的插值函数
f0 = subs(f,'t',x0);          %x0 处的插值
```

例 4-11 第二类三次样条插值应用实例。根据下面的数据点求出其第二类三次样条插值多项式，并计算当 $x=4.3$ 时 y 的值。

x	1	2	3	4	5	6	7	8
y	0.8415	0.9093	0.1411	−0.7568	−0.9589	−0.2794	0.6570	0.9894
y''	−0.8415	−0.9093	−0.1411	0.7568	0.9589	0.2794	−0.6570	−0.9894

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> x=1:8;
>> y=[ 0.8415    0.9093    0.1411   -0.7568   -0.9589   -0.2794    0.6570
    0.9894];
>> [f,f0]=ThrSample2(x,y,-0.8415,-0.9894,4.3)
f =
(-3408335435861847/2251799813685248*t+23858348051032929/450359962737049
6)*(t-5)^2+(-95009442133087537/9007199254740992+8637222012098867/4503599627
370496*t)*(t-4)^2+(-2909697190512605/4503599627370496*t+2909697190512605/11
25899906842624)*(5-t)^2-(6087063306293405/4503599627370496-1217412661258681
/4503599627370496*t)*(t-4)^2
%计算 x=4.3 时的插值输出值 f0
f0 =   -0.9125
```

表格中的数据点是按 $y=\sin x$ 给出的，而 $\sin 4.3=-0.9162$。

（3）对于第三类边界条件，m_i 满足的方程组如下所示：

$$\begin{cases} m_0 = m_N \\ \dfrac{4m_0}{h_0}+\dfrac{2m_1}{h_0}+\dfrac{2m_{N-1}}{h_{N-1}}+\dfrac{4m_N}{h_{N-1}}=\dfrac{6(y_N-y_{N-1})}{h_{N-1}^2}+\dfrac{6(y_1-y_0)}{h_0^2} \\ \lambda_i m_{i-1}+2m_i+\mu_i m_{i+1}=c_i\,(i=1,2,\cdots,N-1) \end{cases}$$

写成矩阵形式为：

$$\begin{bmatrix} 2 & \mu_1 & & & \lambda_1 \\ \lambda_2 & 2 & \mu_2 & & \\ & \lambda_3 & \ddots & \ddots & \\ & & \ddots & 2 & \mu_{N-1} \\ \mu_N & & & \lambda_N & 2 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_{N-1} \\ m_N \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{N-1} \\ c_N \end{bmatrix}$$

各参数的意义如下：

$$\begin{cases} \lambda_N = \dfrac{h_0}{h_{N-1}+h_0}, \mu_N = 1-\lambda_N \\ c_N = \dfrac{3(h_{N-1}\dfrac{y_1-y_0}{h_0} + h_0\dfrac{y_N-y_{N-1}}{h_{N-1}})}{h_{N-1}+h_0} \\ \lambda_i = \dfrac{h_i}{h_i+h_{i-1}}, \mu_i = \dfrac{h_{i-1}}{h_i+h_{i-1}} \\ c_i = \dfrac{3\lambda_i(y_i-y_{i-1})}{h_{i-1}} + \dfrac{3\mu_i(y_{i+1}-y_i)}{h_i} \end{cases}, (i=1,2,\cdots,N-1)$$

在 MATLAB 中编程实现的第三类三次样条插值法函数为：ThrSample3

功能：求已知数据点的第三类三次样条插值多项式及其插值点处的值

调用格式：[f,f0] = ThrSample3 (x,y ,x0)

其中，x：　已知数据点的 x 坐标向量；

y：　已知数据点的 y 坐标向量；

x0：　插值点的 x 坐标；

f：　求得的三次样条插值多项式；

f0：　求得的 x0 处的插值。

在 MATLAB 中实现第三类三次样条插值法的代码如下所示：

```
function [f,f0] = ThrSample3 (x,y,x0)
%求已知数据点的第三类三次样条插值多项式及其插值点处的值
%已知数据点的 x 坐标向量：x
%已知数据点的 y 坐标向量：y
%插值点的 x 坐标：x0
%求得的第三类三次样条插值多项式：f
%求得的 x0 处的插值：f0
syms t;
f = 0.0;
f0 = 0.0;
if(length(x) == length(y))
   n = length(x);
else
   disp('x 和 y 的维数不相等！');
   return;
```

```
end                    %维数检查
for i=1:n
    if(x(i)<=x0)&& (x(i+1)>=x0)
        index = i;
        break;
    end
end                              %找到 x0 所在区间
for i=1:n
    if(x(i)<=x0)&& (x(i+1)>=x0)
        index = i;
        break;
    end
end                              %找到 x0 所在区间
A = diag(2*ones(1,n-1));        %求解 m 的系数矩阵
h0 = x(2)-x(1);
h1 = x(3)-x(2);
he = x(n)-x(n-1);
A(1,2) = h0/(h0+h1);
A(1,n-1) = 1 - A(1,2);
A(n-1,1) = he/(h0+he);
A(n-1,n-2) = 1 - A(n-1,1);
c = zeros(n-1,1);
c(1) = 3* A(1,n-1)*(y(2)-y(1))/h0 + 3* A(1,2)*(y(3)-y(2))/h1;
for i=2:n-2
    u = (x(i)-x(i-1))/(x(i+1)-x(i-1));
    lamda = (x(i+1)-x(i))/(x(i+1)-x(i-1));
    c(i) = 3*lamda*(y(i)-y(i-1))/(x(i)-x(i-1))+ ...
        3*u*(y(i+1)-y(i))/(x(i+1)-x(i));
    A(i, i+1) = u;
    A(i, i-1) = lamda;      %形成系数矩阵及向量 c
end
c(n-1) = 3*( he*(y(2)-y(1))/h0+h0*( y(n)-y(n-1))/he)/(h0+he);
m = zeros(n,1);
[m(2:n),Q,R] = qrxq(A,c);            %用 qr 分解法求解方程组
m(1) = m(n);
h = x(index+1) - x(index);    %x0 所在区间长度
f = y(index)*(2*(t-x(index))+h)*(t-x(index+1))^2/h/h/h + ...
        y(index+1)*(2*(x(index+1)-t)+h)*(t-x(index))^2/h/h/h + ...
        m(index)*(t-x(index))*(x(index+1)-t)^2/h/h - ...
        m(index+1)*(x(index+1)-t)*(t-x(index))^2/h/h;
                          %x0 所在区间的插值函数
f0 = subs(f,'t',x0);          %x0 处的插值
```

例 4-12 第三类三次样条插值应用实例。根据下面的数据点求出其第三类三次样条插值多项式，并计算当 $x=4.3$ 时 y 的值。

x	0	1	2	3	4	5	6.2832
y	0	0.8415	0.9093	0.1411	−0.7568	−0.9589	0

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> x=[0 1   2   3   4   5   6.2832];
>> y=[ 0  0.8415   0.9093   0.1411   -0.7568   -0.9589  0];
>> [f,f0]=ThrSample3(x,y,4.3)
f =
(-946/625*t+3311/625)*(t-5)^2+(-105479/10000+9589/5000*t)*(t-4)^2+(-8015178787200365/9007199254740992*t+8015178787200365/2251799813685248)*(5-t)^2-(-4250778296125375 5/9007199254740992+8501556592250751/9007199254740992*t)*(t-4)^2
%计算 x=4.3 时的插值输出值 f0
f0 =   -0.8718
```

表格中的数据点是按 $y=\sin x$ 给出的，而 $\sin 4.3=-0.9162$。x 这样选取是为了满足插值条件，但是由于计算误差，使得 $\sin 6.2832$ 不精确等于 0，因此结果有一定的误差。

4.7.3 B 样条插值

B 样条插值也称为等距节点三次样条插值。B 样条函数是最基本的样条函数。它是基于磨光函数而形成的。磨光函数也就是把函数作一次不定积分，然后进行一次步长为 h 的对称差商运算，得到的结果就是一个磨光函数。利用磨光函数对于插值节点进行平滑处理，这样得到的结果就是一个基本的样条函数。如果使用 k 次的磨光函数的话，得到的插值样条函数就为 k 次 B 样条函数。

如果节点之间是等距的，设 $x_i=a+ih(i=0,1,\cdots,n,h=\dfrac{b-a}{n})$，则插值函数可写成：$p(x)=\sum\limits_{j=-1}^{n+1}c_j\Omega_3(\dfrac{x-x_0}{h}-j)$。

其中 $\Omega_3(\dfrac{x-x_0}{h}-j)$ 为 B 样条。根据不同的边界条件可得出不同的系数，如上小节一样，第一种条件下的系数求法如下所示：

$$\begin{bmatrix}4 & 2 & & & \\ 1 & 4 & \ddots & & \\ & \ddots & \ddots & 1 & \\ & & 1 & 4 & 1 \\ & & & 2 & 4\end{bmatrix}\begin{bmatrix}c_0 \\ c_1 \\ \vdots \\ c_{n-1} \\ c_n\end{bmatrix}=\begin{bmatrix}6y_0+2hy_0' \\ 6y_1 \\ \vdots \\ 6y_{n-1} \\ 6y_n-2hy_n'\end{bmatrix}$$

求解出 $c_0,c_1,\cdots,c_n$ 后，再用下面的式子求出 c_{-1} 和 c_{n+1}：

$$\begin{cases}c_{-1}=c_1-2hy_0' \\ c_{n+1}=c_{n-1}-2hy_n'\end{cases}$$

于是插值函数为 $p(x)=\sum\limits_{j=-1}^{n+1}c_j\Omega_3(\dfrac{x-x_0}{h}-j)$。

在 MATLAB 中编程实现的第一类 B 样条插值函数为：BSample

功能：求已知数据点的第一类 B 样条的插值

调用格式：f0 = BSample(a,b,n,y,y_1,y_N,x0)

其中，	a：	区间左端点；
	b：	区间右端点；
	y：	已知数据点的 y 坐标向量；
	n：	区间等分数；
	y_1：	左端点的一阶导数；
	y_N：	右端点的一阶导数；
	x0：	插值点的 x 坐标；
	f0：	求得的 x0 处的插值。

在 MATLAB 中实现第一类 B 样条插值的代码如下所示：

```
function f0 = BSample(a,b,n,y,y_1,y_N,x0)
%求已知数据点的第一类 B 样条插值
%区间左端点: a
%区间右端点: b
%区间等分数: n
%已知数据点的 y 坐标向量: y
%左端点的一阶导数;: y_1
%右端点的一阶导数;: y_N
%插值点的 x 坐标: x0
%x0 处的插值: f0
f0 = 0.0;
h = (b-a)/n;
c = zeros(n+3,1);
b = zeros(n+1,1);
for i=0:n-1
    if(a+i*h<=x0) && (a+i*h+h>=x0)
        index = i;
        break;
    end
end                              %找到 x0 所在区间
A = diag(4*ones(n+1,1));
I = eye(n+1,n+1);
AL = [I(2:n+1,:);zeros(1,n+1)];
AU = [zeros(1,n+1);I(1:n,:)];
A = A+AL+AU;                        %形成系数矩阵
for i=2:n
    b(i,1) = 6*y(i);
end
b(1) = 6*y(1)+2*h*y_1;
b(n+1) = 6*y(n+1)-2*h*y_N;
d = followup(A,b);              %用追赶法求出系数
c(2:n+2) = d;
c(1) = c(2) - 2*h*y_1;         %c(-1)
c(n+3) = c(3)+2*h*y_N;         %c(n+1)
x1 = (a+index*h-h-x0)/h;
m1 = c(index+1)*(-((abs(x1))^3)/6+(x1)^2-2*abs(x1)+4/3);
```

```
x2 = (a+index*h-x0)/h;
m2 = c(index+2)*((abs(x2))^3/2-(x2)^2+2/3);
x3 = (a+index*h+h-x0)/h;
m3 = c(index+3)*((abs(x3))^3/2-(x3)^2+2/3);
x4 = (a+index*h+2*h-x0)/h;
m4 = c(index+4)*(-((abs(x4))^3)/6+(x4)^2-2*abs(x4)+4/3);
f0 = m1+m2+m3+m4;              %求出插值
```

例 4-13 第一类 B 样条插值应用实例。x 为区间[0,6.2832]的等分节点，等分份数为 10，而 $y=\sin x$。求出当 $x=2.8$ 时，第一类 B 样条插值的 y 值。

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> x=0:6.2832/10:6.2832;
>> y=sin(x);
>> f0 = BSample(0,6.2832,10,y,1,cos(6.2832),2.8) %计算 x=2.8 时的插值输出值
f0 =    0.3352
```

而 $\sin 2.8 = 0.3350$，从插值的结果来看，误差不超过 0.06%。

B 样条的定义如下：

$$\Omega_3=\begin{cases}0 & |x|\geqslant 2\\ 0.5|x|^3-x^2+2/3 & |x|\leqslant 1\\ -|x|^3/6+x^2-2|x|+4/3, & 1<|x|<2\end{cases}$$

对于第二类 B 样条，其系数求解公式为：

$$\begin{bmatrix}4 & 1 & & & \\ 1 & 4 & \ddots & & \\ & \ddots & \ddots & 1 & \\ & & 1 & 4 & 1\\ & & & 1 & 4\end{bmatrix}\begin{bmatrix}c_1\\ c_2\\ \vdots\\ c_{n-2}\\ c_{n-1}\end{bmatrix}=\begin{bmatrix}6y_1-y_0+\dfrac{h^2}{6}y_0''\\ 6y_2\\ \vdots\\ 6y_{n-2}\\ 6y_{n-1}-y_n+\dfrac{h^2}{6}y_n''\end{bmatrix}$$

求解出 $c_1,c_2,\cdots,c_{n-1}$ 后，再用下面的式子求出 c_{-1}，c_0，c_n 和 c_{n+1}：

$$\begin{cases}c_{-1}=2c_0-c_1+h^2y_0''\\ c_0=y_0-\dfrac{h^2}{6}y_0''\\ c_n=y_n-\dfrac{h^2}{6}y_n''\\ c_{n+1}=2c_n-c_{n-1}+h^2y_n''\end{cases}$$

程序的求解过程大体不变，只不过在用追赶法求解系数 c_i 的时候稍微改一下矩阵 A 和向量 b 即可。

对于第三类 B 样条，其系数求解公式为：

$$\begin{bmatrix} 4 & 1 & & & \\ 1 & 4 & \ddots & & \\ & \ddots & \ddots & 1 & \\ & & 1 & 4 & 1 \\ & & & 1 & 4 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \\ c_n \end{bmatrix} = \begin{bmatrix} 6y_1 \\ 6y_2 \\ \vdots \\ 6y_{n-1} \\ 6y_0 \end{bmatrix}$$

而 $c_{n+1}=c_1, c_0=c_n, c_{-1}=c_{n-1}$。其程序更简单，读者参考第一类 B 样条的代码就可以很快写出来。

4.8 有理分式插值

所谓有理分式插值，就是寻求如下形式的有理分式函数：

$$R_{m.n}=\frac{a_0+a_1x+\cdots+a_mx^m}{b_0+b_1x+\cdots+b_nx^n}$$

满足 $R_{m.n}(x_i)=f(x_i)$。

在已知 $f(x_i)$ 的情况下，可以通过以下两种算法来得到 $R_{m.n}$：

1. 反差商-连分式算法

定义一阶反差商的计算公式如下所示：

$$f[x_0,x_m]^{-1}=\frac{x_m-x_0}{f(x_m)-f(x_0)}$$

一般情况下，k 阶反差商定义为：

$$f[x_0,x_1,\cdots,x_{k-1},x_m]^{-1}=\frac{x_m-x_{k-1}}{f[x_0,x_1,\cdots,x_{k-2},x_m]-f[x_0,x_1,\cdots,x_{k-1}]}$$

显然，反差商就是均差的倒数。仿照均差表，可构造如表 4-4 所示的反差商表。

表 4-4　反差商计算表格

	一阶均差	二阶均差	三阶均差	…	n 阶均差
$f(x_0)$					
$f(x_1)$	$f[x_0,x_1]^{-1}$				
$f(x_2)$	$f[x_0,x_2]^{-1}$	$f[x_0,x_1,x_2]^{-1}$			
$f(x_3)$	$f[x_0,x_3]^{-1}$	$f[x_0,x_1,x_3]^{-1}$	$f[x_0,x_1,x_2,x_3]^{-1}$		
…	…	…	…		
$f(x_n)$	$f[x_0,x_n]^{-1}$	$f[x_0,x_1,x_n]^{-1}$	$f[x_0,x_1,x_2,x_n]^{-1}$	…	$f[x_0,x_1,\cdots,x_n]^{-1}$

则最终的插值公式可表示为：

$$R_{m.n}=f(x_0)+\cfrac{x-x_0}{f[x_0,x_1]^{-1}+\cfrac{x-x_1}{f[x_0,x_1,x_2]^{-1}+\cfrac{x-x_2}{\ddots\cfrac{x-x_{n-1}}{f[x_0,x_1,x_2,\cdots,x_n]^{-1}}}}}$$


对给定的数据 (x_i, y_i) 组，并不一定存在对应的有理分式插值函数。

在 MATLAB 中编程实现的有理分式形式的反差商插值法函数为：DCS
功能：用反差商算法求已知数据点的有理分式形式的插值分式
调用格式：f = DCS (x,y)或 f = DCS (x,y,x0)
其中，x：　已知数据点的 x 坐标向量；
y：　已知数据点的 y 坐标向量；
x0：　插值点的 x 坐标；
f：　求得的有理分式形式的插值分式或是在 x0 处的插值。

在 MATLAB 中实现有理分式插值（反差商算法）的代码如下所示：

```
function f = DCS(x,y,x0)
%用反差商算法求已知数据点的有理分式形式的插值分式
%已知数据点的 x 坐标向量：x
%已知数据点的 y 坐标向量：y
%插值点的 x 坐标：x0
%求得的有理分式形式的插值分式或是在 x0 处的插值：f
syms t;
if(length(x) == length(y))
    n = length(x);
    c(1:n) = 0.0;
else
    disp('x 和 y 的维数不相等！');
    return;
end
c(1) = y(1);
for(i=1:n-1)
    for(j=i+1:n)
        y1(j) = (x(j)-x(i))/(y(j)-y(i));      %计算反差商
    end
    c(i+1) = y1(i+1);
    y = y1;
end
f = c(n);
for(i=1:n-1)
    f = c(n-i) + (t-x(n-i))/f;               %插值公式的连分式递推算法
    f = vpa(f,6);
    if(i==n-1)
        if(nargin == 3)
            f = subs(f,'t',x0);
        else
            f = vpa(f,6);
        end
    end
end;
```

例 4-14　有理分式插值法（反差商法）应用实例。根据下面的数据点，用有理分式

的反差商形式进行插值，并计算当 $x=1.44$ 时 y 的值。

x	1	1.2	1.4	1.6	1.8
y	1	1.0954	1.1832	1.2649	1.3416

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> x=1:0.2:1.8;
>> y=[1  1.0954  1.1832  1.2649 1.3416];
>> f=DCS(x,y)
f =
1.+(t-1.)/(2.09545+(t-1.20000)/(2.27866+(t-1.40000)/(1.83429+.383649*t)))
>> f=DCS(x,y,1.44)   %计算 x=1.44 时的插值输出值 f
f = 1.2000
```

表格中的数据点是按 $y=\sqrt{x}$ 给出的，从插值结果看，偏差也不大，如果增加插值点的个数，精度会更高。

2. Neville 迭代算法

由 Neville 迭代算法得到 $R_{m,n}$ 的迭代公式如下所示：

$$N_{i,k}(x)=\begin{cases}0,k=0\\ f(x_i),k=1\\ N_{i,k-1}(x)+\dfrac{N_{i,k-1}(x)-N_{i-1,k-1}(x)}{\dfrac{x-x_{i-k}}{x-x_i}[1-\dfrac{N_{i,k-1}(x)-N_{i-1,k-1}(x)}{N_{i,k-1}(x)-N_{i-1,k-2}(x)}]},k\geqslant 2\end{cases},1\leqslant k<i,i=1,2,\cdots,n$$

将算法列成表格如下所示：

i	$N_{i,1}$	$N_{i,2}$	$N_{i,3}$	…
1	$f(x_1)$			
2	$f(x_2)$	$N_{2,2}$		
3	$f(x_3)$	$N_{3,2}$	$N_{3,3}$	
4	$f(x_4)$			
…	…	…	…	

最终的 $N_{i,i-1}$ 就是所要的插值分式。



对给定的数据 (x_i,y_i) 组，用 Neville 算法不一定能得到有理分式插值函数，在迭代过程中可能出现分母为零的情况。

在 MATLAB 中编程实现的有理分式形式的 Neville 插值法函数为：Neville

功能：用 Neville 算法求已知数据点的有理分式形式的插值分式

调用格式：f＝Neville (x,y)或 f＝Neville (x,y,x0)

其中，x：　　已知数据点的 x 坐标向量；

　　　y：　　已知数据点的 y 坐标向量；

x0:	插值点的 x 坐标;
f:	求得的有理分式形式的插值分式或是在 x0 处的插值。

在 MATLAB 中实现有理分式插值（Neville 算法）的代码如下所示:

```
function f = Neville(x,y,x0)
%用 Neville 算法求已知数据点的有理分式形式的插值分式
%已知数据点的 x 坐标向量: x
%已知数据点的 y 坐标向量: y
%插值点的 x 坐标: x0
%求得的有理分式形式的插值分式或是在 x0 处的插值: f
syms t;
if(length(x) == length(y))
   n = length(x);
else
   disp('x 和 y 的维数不相等! ');
   return;
end
y1(1:n) = t;
for(i=1:n-1)
    for(j=i+1:n)        %递推公式
        if(j==2)         %注意递推公式中的下标范围
            y1(j) = y(j)+(y(j)-y(j-1))/((t-x(j-i))/(t-x(j)))*(1-(y(j)-
y(j-1))/y(j));
        else
            y1(j) = y(j)+(y(j)-y(j-1))/((t-x(j-i))/(t-x(j)))*(1-(y(j)-
y(j-1))/(y(j)-y(j-2)));
        end
    end
    y = y1;
    if(i==n-1)
        if(nargin == 3)
            f = subs(y(n-1),'t',x0);
        else
            f = vpa(y(n-1),6);
        end
    end
end
```

例 4-15 有理分式插值法（Neville 算法）应用实例。根据下面的数据点，用有理分式的 Neville 形式进行插值，并计算当 $x=1.44$ 时 y 的值。

x	1	1.2	1.4	1.6	1.8
y	1	1.0954	1.1832	1.2649	1.3416

解: 在 MATLAB 命令窗口中输入以下命令:

```
>> x=1:0.2:1.8;
>> y=[1  1.0954  1.1832  1.2649  1.3416];
>> f=Neville(x,y);                    %表达式过长，在此不再列出
```

```
>> f= Neville(x,y,1.44)    %计算 x=1.44 时的插值输出值 f
 f =    1.0030
```

表格中的数据点是按 $y=\sqrt{x}$ 给出的，从插值结果看，偏差比较大，一般来说迭代算法的精度很难保证。

4.9 反插值

如果已知插值的函数值，要求对应插值点的 x 值，则可用反插值。构造反插值多项式的算法介绍如下：

❶ 利用表 4-4 计算反差商；

❷ 计算反插值多项式：

$$p(y)=x_0+f[y_0,y_1]^{-1}(y-y_0)+\\ f[y_0,y_1,y_2]^{-1}(y-y_0)(y-y_1)+\cdots+\\ f[y_0,y_1,\cdots,y_n]^{-1}(y-y_0)(y-y_1)\cdots(y-y_n)$$

在 MATLAB 中编程实现的反插值函数为：FCZ

功能：用反差商算法求已知数据点的有理分式形式的插值分式

调用格式：[f ,f0]= FCZ (x,y,y0)

其中，x：　已知数据点的 x 坐标向量；

y：　已知数据点的 y 坐标向量；

y0：　反插值点的 y 坐标；

f：　求得反插值函数表达式；

f0：　反插值点的 x 值。

在 MATLAB 中实现反插值算法的代码如下所示：

```
function [f,x0] = FCZ(x,y,y0)
%用反差商算法求已知数据点的有理分式形式的插值分式
%已知数据点的 x 坐标向量: x
%已知数据点的 y 坐标向量: y
%插值点的 y 坐标: y0
%求得反插值函数表达式: f
%求得反插值点的 x 值: f0
syms t;
if(length(x) == length(y))
   n = length(x);
   c(1:n) = 0.0;
else
   disp('x 和 y 的维数不相等! ');
   return;
end
c(1) = x(1);
```

```
y1 = x;
for(i=1:n-1)
    for(j=i+1:n)
        y2(j) = (y1(j)-y1(i))/(y(j)-y(i));     %反差商
    end
    c(i+1) = y2(i+1);
    y1 = y2;
end
f = c(1);
for(i=1:n-1)
    ff = c(i+1);
    for(j=1:i)        %反插值多项式的项
        ff = ff*(t-y(j));
    end
    f = f + ff;     %反插值多项式
end;
x0 = subs(f,'t',y0);
```

假设 c 是介于 y_0 和 y_1 之间的一个实数，如果被插值的函数为连续函数，则在 x_0 和 x_1 之间存在某个 x_c，使得 $c = f(x_c)$，求 x_c 的另一种算法步骤介绍如下。

❶ 令 $x^0 = x_0 + \dfrac{c - f(x_0)}{f[x_0, x_1]}$；

❷ $x^1 = x_0 + \dfrac{c - f(x_0)}{f[x_0, x_1]} - \dfrac{f[x_0, x_1, x_2]}{f[x_0, x_1]}(x^0 - x_0)(x^0 - x_1)$；

❸ 按下面的迭代公式进行迭代计算：

$$x^k = x^0 - \frac{f[x_0, x_1, x_2]}{f[x_0, x_1]}(x^{k-1} - x_0)(x^{k-1} - x_1)$$

进行迭代，直到 x^k 与 x^{k-1} 之间的插值小于某个给定的精度值。

在 MATLAB 中编程实现的迭代反插值函数为：DFCZ

功能：用迭代法求已知点的反插值点

调用格式：xf = DFCZ (x,y,y0,eps)

其中，x：　已知数据点的 x 坐标向量；

y：　已知数据点的 y 坐标向量；

y0：　反插值点的 y 坐标；

eps：　迭代精度；

xf：　反插值点的 x 值。

在 MATLAB 中实现迭代反插值算法的代码如下所示：

```
function xf = DFCZ(x,y,y0,eps)
%用迭代法求已知点的反插值点
%已知数据点的 x 坐标向量：x
%已知数据点的 y 坐标向量：y
```

```
%插值点的 y 坐标: y0
%迭代精度:   eps
%求得反插值函数表达式: f
%求得反插值点的 x 值: f0
if(length(x) == length(y))
    n = length(x);
    if(nargin == 3)
        eps = 1.0e-6;
    end
else
    disp('x 和 y 的维数不相等! ');
    return;
end
for i=1:n
    if(y(i)<=y0)&& (y(i+1)>=y0)
        index = i;
        break;
    end
end                              %找到 y0 所在区间
tol = 1;
xf0=x(index)+(y0-y(index))*(x(index+1)-x(index))/(y(index+1)-y(index));
xf1 = xf0;
while tol>eps       %迭代找出反插值点
     d1 = (y(index+2)-y(index))/( x(index+2)-x(index));
     d2 = (y(index+1)-y(index))/( x(index+1)-x(index));
     d3 = (d1 - d2)/( x(index+2)-x(index+1));
     xf = xf0 - d3*(xf1-x(index))*(xf1-x(index+1))/d2;
     tol = abs(xf - xf1);
     xf1 = xf;
end
```

例 4-16 反插值应用实例。根据下面的数据点，计算当 $y=1.44$ 时 x 的值。

x	1	1.2	1.4	1.6	1.8
y	1	1.0954	1.1832	1.2649	1.3416

解: 在 MATLAB 命令窗口中输入以下命令:

```
>> x=1:0.2:1.8;
>> y=[1  1.0954  1.1832  1.2649  1.3416];
>> [f,x0] =FCZ(x,y,1.44)
   f =-616680776470671/562949953421312+1179630729891983/562949953421312*t+
(281474976710651/281474976710656*t-281474976710651/281474976710656)*(t-1/5*
30^(1/2))+(5437323133546911/39614081257132168796771975168*t-543732313354691
1/39614081257132168796771975168)*(t-1/5*30^(1/2))*(t-1/5*35^(1/2))+(-419969
3456240639/4951760157141521099596496896*t+4199693456240639/4951760157141521
099596496896)*(t-1/5*30^(1/2))*(t-1/5*35^(1/2))*(t-2/5*10^(1/2))
   %计算 y=1.44 时的 x 值
```

```
x0 =    2.0736
```

表格中的数据点是按 $y=\sqrt{x}$ 给出的，而当 $y=1.44$ 时，$1.44^2=2.0736$，从插值结果看，是很准确的，而且更重要的是此例求的是外插点的值。

再看看迭代反插值的应用，题目仍然如例 4-15 所示：

```
>> x0=DFCZ(x,y,1.1)
x0 =1.2101
```

当 $y=1.1$ 时，$1.1^2=1.21$，可见结果也是相当精确的。

一般来说，反插值可以用来求函数的根，例如要求 $x^3-3x^2+x+1=0$ 在区间[0.5,1.4]的一个根，则可以用以下的方法求解：

```
>> x=0.5:0.16:1.4;
>> for i=1:6
     y(i)=x(i)^3-3*x(i)^2+x(i)+1;
    end
>> [f,x0] =FCZ(x,y,0)
f =574/523-2500/3661*t+(-8609847242294533/36028797018963968*t+602689306
96061731/288230376151711744)*(t-5770876533715533/9007199254740992)+(-678135
9464980797/36028797018963968*t+47469516254865579/288230376151711744)*(t-577
0876533715533/9007199254740992)*(t-1595030872826553/4503599627370496)+(-672
9573403253353/72057594037927936*t+47107013822773471/576460752303423488)*(t-
5770876533715533/9007199254740992)*(t-1595030872826553/4503599627370496)*(t
-900539781489 01/2251799813685248)+(-8666552771671405/144115188075855872*t+6
0665869401699835/1152921504606846976)*(t-5770876533715533/9007199254740992)
*(t-1595030872826553/4503599627370496)*(t-900539781489 01/2251799813685248)*
(t+312162504571559/1125899906842624)
    x0 =    1.0000
```

经过验证可以得出 $x=1.0$ 确实是方程 $x^3-3x^2+x+1=0$ 的一个根，当然区间要适当细分才能得到精度比较好的根。

4.10 二维插值

前面讲述的都是一维插值，即节点为一维变量，插值函数是一元函数（曲线）。若节点是二维的，插值函数就是二元函数，即曲面。例如在某区域测量了若干点（节点）的高程（节点值），为了画出较精确的等高线图，就要先插入更多的点（插值点），计算这些点的高程（插值）。

4.10.1 分片双线性插值

双线性插值是由一片一片的二次曲面空间构成。

定义矩形网格：

$$a=x_0<x_1<\cdots<x_n=b$$
$$c=y_0<y_1<\cdots<y_m=d$$

在其上给出函数值：

$$z(x_i, y_j) = z_{ij} \quad 0 \leqslant i \leqslant n, 0 \leqslant j \leqslant m$$

在矩形网格上的某个小片 $[x_{i-1}, x_i] \times [y_{j-1}, y_j]$ 上的双线性插值函数为：

$$L(x, y) = a + bx + cy + dxy$$

其中，

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 1 & x_{i-1} & y_{j-1} & x_{i-1}y_{j-1} \\ 1 & x_i & y_j & x_i y_j \\ 1 & x_{i-1} & y_j & x_{i-1}y_j \\ 1 & x_i & y_{j-1} & x_i y_{j-1} \end{bmatrix}^{-1} \begin{bmatrix} z_{i-1,j-1} \\ z_{i,j} \\ z_{i-1,j} \\ z_{i,j-1} \end{bmatrix}$$

在 MATLAB 中编程实现的双线性插值函数为：DL

功能：用双线性插值求已知点的插值

调用格式：fz = DL (x,y,Z,x0,y0)

其中，x：　已知数据点的 x 坐标向量；

y：　已知数据点的 y 坐标向量；

Z：　已知数据点的 z 坐标矩阵；

x0：　插值点的 x 坐标；

y0：　插值点的 y 坐标；

fz：　插值点的 z 值。

在 MATLAB 中实现双线性插值的代码如下所示：

```
function fz = DL(x,y,Z,x0,y0,eps)
%用双线性插值求已知点的插值
%已知数据点的 x 坐标向量：x
%已知数据点的 y 坐标向量：y
%已知数据点的 z 坐标矩阵：Z
%插值点的 x 坐标：x0
%插值点的 y 坐标：y0
%求得插值点的 z 值：fz
n = length(x);
m = length(y);
for i=1:n
    if(x(i)<=x0)&& (x(i+1)>=x0)
        index_x = i;
        break;
    end
end                              %找到 x0 所在区间
for i=1:m
    if(y(i)<=y0)&& (y(i+1)>=y0)
        index_y = i;
        break;
    end
end                              %找到 y0 所在区间
```

```
A = [1 x(index_x) y(index_y) x(index_x)* y(index_y);
     1 x(index_x+1) y(index_y+1) x(index_x+1)* y(index_y+1);
     1 x(index_x) y(index_y+1) x(index_x)* y(index_y+1);
     1 x(index_x+1) y(index_y) x(index_x+1)* y(index_y)];
iA = inv(A);
B = iA*[Z(index_x,index_y); Z(index_x+1,index_y+1); Z(index_x,index_y+1);
        Z(index_x+1,index_y)];
fz = [1 x0 y0 x0*y0]*B;
```

例 4-17 分片双线性插值应用实例。根据下面的数据点（表格中的值是 z 值，按照 $z=\sin x\cos y$ 给出）计算当 $x=2.08, y=3.77$ 时 z 的值。

y \ x	1	2	3	4	5	6
2	-0.3502	-0.833	-0.55	0.2387	0.808	0.6344
3	-0.3784	-0.9002	-0.5944	0.2579	0.8731	0.6855
4	-0.0587	-0.1397	-0.0922	0.04	0.1355	0.1064
5	0.3149	0.7492	0.4947	-0.2147	-0.7267	-0.5706
6	0.3991	0.9493	0.6268	-0.272	-0.9207	-0.7229
7	0.1163	0.2766	0.1826	-0.0793	-0.2683	-0.2107

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> x=1:6;
>> y=2:7;
>> z=sin(x)'*cos(y)
z =
   -0.3502   -0.8330   -0.5500    0.2387    0.8080    0.6344
   -0.3784   -0.9002   -0.5944    0.2579    0.8731    0.6855
   -0.0587   -0.1397   -0.0922    0.0400    0.1355    0.1064
    0.3149    0.7492    0.4947   -0.2147   -0.7267   -0.5706
    0.3991    0.9493    0.6268   -0.2720   -0.9207   -0.7229
    0.1163    0.2766    0.1826   -0.0793   -0.2683   -0.2107
>> fz=DL(x,y,z,2.08,3.77)  %计算 x=2.08,y=3.77 时的插值输出值 fz
fz =   -0.6198
>> sin(2.08)*cos(3.77)  %计算真实值
ans =   -0.7063
```

由于是线性插值，因此结果显然不会太好，这从例题的结果中也可以看出。

4.10.2 二元三点拉格朗日插值

在矩形网格某个小片 $[x_{i-1},x_i]\times[y_{j-1},y_j]$ 上的二元三点拉格朗日函数为：

$$Q(x,y)=\sum_{i=p}^{p+2}\sum_{j=q}^{q+2}\prod_{\substack{k=p\\k\neq i}}^{p+2}\left(\frac{x-x_k}{x_i-x_k}\right)\prod_{\substack{l=q\\l\neq j}}^{q+2}\left(\frac{y-y_l}{y_j-y_l}\right)z_{ij}$$

其中，x_p, x_{p+1}, x_{p+2} 和 y_q, y_{q+1}, y_{q+2} 分别为网格中最靠近插值点 $(x0, y0)$ 的 x 方向的坐标和 y 方向的坐标。

在 MATLAB 中编程实现的二元三点拉格朗日插值函数为：DTL

功能：用二元三点拉格朗日插值求已知点的插值

调用格式：[f,fz]= DTL (x,y,Z, x0,y0)

其中，x：　已知数据点的 x 坐标向量；

y：　已知数据点的 y 坐标向量；

Z：　已知数据点的 z 坐标矩阵；

x0：　插值点的 x 坐标；

y0：　插值点的 y 坐标；

f：　插值多项式；

fz：　插值点的 z 值。

在 MATLAB 中实现二元三点拉格朗日插值的代码如下所示：

```
function fz = DTL(x,y,Z,x0,y0)
%用二元三点拉格朗日插值求已知点的插值
%已知数据点的 x 坐标向量：x
%已知数据点的 y 坐标向量：y
%已知数据点的 z 坐标矩阵：Z
%插值点的 x 坐标：x0
%插值点的 y 坐标：y0
%插值多项式：        f
%求得插值点的 z 值：fz
syms s t;
f = 0.0;
n = length(x);
m = length(y);
for i=1:n
    if(x(i)<=x0)&& (x(i+1)>=x0)
        index_x = i;
        break;
    end
end                          %找到 x0 所在区间
for i=1:m
    if(y(i)<=y0)&& (y(i+1)>=y0)
        index_y = i;
        break;
    end
end                          %找到 y0 所在区间
if index_x == 1
    cx(1:3) = index_x:(index_x+2);
else
    if index_x == n-1
        cx(1:3) = (index_x-1):(index_x+1);
    else
        if abs(x(index_x-1)-x0)>abs(x(index_x+2)-x0)
            cx(1:3) = (index_x):(index_x+2);
        else
            cx(1:3) = (index_x-1):(index_x+1);
        end
```

```
        end
    end                         %找到离 x0 最近的三个 x 坐标
    if index_y == 1
        cy(1:3) = index_y:(index_y+2);
    else
        if index_y == m-1
            cy(1:3) = (index_y-1):(index_y+1);
        else
            if abs(y(index_y-1)-y0)>=abs(y(index_y+2)-y0)
                cy(1:3) = (index_y):(index_y+2);
            else
                cy(1:3) = (index_y-1):(index_y+1);
            end
        end
    end                         %找到离 y0 最近的三个 y 坐标
    for i=1:3
        i1 = mod(i+1,3);
        if(i1 == 0)
            i1 = 3;
        end
        i2 = mod(i+2,3);
        if(i2 == 0)
            i2 = 3;
        end
        for j=1:3
          j1 = mod(j+1,3);
          if(j1 == 0)
              j1 = 3;
          end
          j2 = mod(j+2,3);
          if(j2 == 0)
              j2 = 3;
          end
          f = f+Z(cx(i),cy(j))*((t-x(cx(i1)))*(t-x(cx(i2)))/(x(cx(i))-
x(cx(i1)))/(x(cx(i))-x(cx(i2))))* ...
               (s-y(cy(j1)))*(s-y(cy(j2)))/(y(cy(j))-y(cy(j1)))/(y(cy(j))-
y(cy(j2))); %插值多项式
            end
    end
    fz = subs(f,'[t s]',[x0 y0]);
```

例 4-18　二元三点拉格朗日插值应用实例。根据下面的数据点(表格中的值是 z 值，按照 $z=\sin x\cos y$ 给出计)算当 $x=1.3, y=2.7$ 时 z 的值。

y \ x	1	2	3	4	5	6
2	-0.3502	-0.833	-0.55	0.2387	0.808	0.6344
3	-0.3784	-0.9002	-0.5944	0.2579	0.8731	0.6855
4	-0.0587	-0.1397	-0.0922	0.04	0.1355	0.1064
5	0.3149	0.7492	0.4947	-0.2147	-0.7267	-0.5706
6	0.3991	0.9493	0.6268	-0.272	-0.9207	-0.7229
7	0.1163	0.2766	0.1826	-0.0793	-0.2683	-0.2107

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> x=1:6;
>> y=2:7;
>> z=sin(x)'*cos(y)
>> [f,fz]=DTL(x,y,z,1.3,2.7)
f = -6308200795821975/72057594037927936*(t-2)*(t-3)*(s-3)*(s-4)+3751723
494241469/9007199254740992*(t-2)*(t-3)*(s-4)*(s-2)-4954159021762373/3602879
7018963968*(t-2)*(t-3)*(s-2)*(s-3)+6816670871723695/36028797018963968*(t-3)
*(t-1)*(s-3)*(s-4)-506766213729585/562949953421312*(t-3)*(t-1)*(s-4)*(s-2)+
5353487086191315/18014398509481984*(t-3)*(t-1)*(s-2)*(s-3)-8463401478808105
/576460752303423488*(t-1)*(t-2)*(s-3)*(s-4)+78648470848139/1125899906842624
*(t-1)*(t-2)*(s-4)*(s-2)-6646750499572631/288230376151711744*(t-1)*(t-2)*(s
-2)*(s-3)
%计算 x=1.3, y=2.7 时的插值输出值 fz
fz =   -0.8674
>> sin(1.3)*cos(2.7) %计算真实值
ans =   -0.8711
```

从例题的结果可以看出，插值结果较好。

4.10.3 分片双三次埃尔米特插值

在矩形网格某个小片 $[x_{i-1}, x_i]\times[y_{j-1}, y_j]$ 上的双三次埃尔米特插值函数为：

$$H_{ij}(x,y) = [(1-\Delta_x)^2(1+2\Delta_x) \quad \Delta_x(1-\Delta_x)^2 \quad \Delta_x^2(3-2\Delta_x) \quad \Delta_x^2(\Delta_x-1)] \cdot [C] \cdot \begin{bmatrix} (1-\Delta_y)^2(1+2\Delta_y) \\ \Delta_y(1-\Delta_y)^2 \\ \Delta_y^2(3-2\Delta_y) \\ \Delta_y^2(\Delta_y-1) \end{bmatrix}$$

其中 $\Delta_x = \dfrac{x0-x_{i-1}}{x_i - x_{i-1}}, \Delta_y = \dfrac{y0-y_{i-1}}{y_i-y_{i-1}}$，而 $(x0, y0)$ 为插值点：

$$[C] = \begin{bmatrix} z_{i-1,j-1} & z_{i-1,j} & (\frac{\partial z}{\partial y})_{i-1,j-1} & (\frac{\partial z}{\partial y})_{i-1,j} \\ z_{i,j-1} & z_{i,j} & (\frac{\partial z}{\partial y})_{i,j-1} & (\frac{\partial z}{\partial y})_{i,j} \\ (\frac{\partial z}{\partial x})_{i-1,j-1} & (\frac{\partial z}{\partial x})_{i-1,j} & (\frac{\partial^2 z}{\partial x \partial y})_{i-1,j-1} & (\frac{\partial^2 z}{\partial x \partial y})_{i-1,j} \\ (\frac{\partial z}{\partial x})_{i,j-1} & (\frac{\partial z}{\partial x})_{i,j} & (\frac{\partial^2 z}{\partial x \partial y})_{i,j-1} & (\frac{\partial^2 z}{\partial x \partial y})_{i,j} \end{bmatrix}$$

在 MATLAB 中编程实现的分片双三次埃尔米特插值函数为：DH

功能：用分片双三次埃尔米特插值求插值点的 z 坐标

调用格式：fz = DH (x,y,Z, zx,zy,zxy,x0,y0)

其中，x：　　已知数据点的 x 坐标向量；

y:	已知数据点的 y 坐标向量;
Z:	已知数据点的 z 坐标矩阵;
zx:	z 对 x 的偏导数矩阵;
zy:	z 对 y 的偏导数矩阵;
zxy:	z 对 x 和 y 的偏导数矩阵;
x0:	插值点的 x 坐标;
y0:	插值点的 y 坐标;
fz:	插值点的 z 值。

在 MATLAB 中实现分片双三次埃尔米特插值的代码如下所示:

```
function fz = DH(x,y,x0,y0,zx,zy,zxy)
%用分片双三次埃尔米特插值求插值点的 z 坐标
%已知数据点的 x 坐标向量: x
%已知数据点的 y 坐标向量: y
%已知数据点的 z 坐标矩阵: Z
z 对 x 的偏导数矩阵: zx
z 对 y 的偏导数矩阵: zy
z 对 x 和 y 的偏导数矩阵:  zxy
%插值点的 x 坐标: x0
%插值点的 y 坐标: y0
%求得插值点的 z 值: fz
n = length(x);
m = length(y);
for i=1:n
    if(x(i)<=x0)&& (x(i+1)>=x0)
        index_x = i;
        break;
    end
end                          %找到 x0 所在区间
for i=1:m
    if(y(i)<=y0)&& (y(i+1)>=y0)
        index_y = i;
        break;
    end
end                          %找到 y0 所在区间
hx = x(index_x+1) - x(index_x);
hy = y(index_y+1) - y(index_y);
tx = (x0 - x(index_x))/hx;          %插值坐标归一化
ty = (y0 - y(index_y))/hy;          %插值坐标归一化
Hl = [(1-tx)^2*(1+2*tx)  tx*tx*(3-2*tx) tx*(1-tx)^2  tx*tx*(tx-1)];  %左向量
Hr = [(1-ty)^2*(1+2*ty); ty*ty*(3-2*ty); ty*(1-ty)^2 ; ty*ty*(ty-1)];  %右向量
C = [Z(index_x, index_y) Z(index_x,index_y+1) zy(index_x, index_y) ...
zy(index_x, index_y+1);
Z(index_x+1, index_y) Z(index_x+1,index_y+1) zy(index_x+1, index_y) ...
zy(index_x+1, index_y+1);
zx(index_x, index_y) zy(index_x, index_y+1) zxy(index_x, index_y) ...
```

```
zxy(index_x, index_y+1);
zx(index_x+1, index_y) zy(index_x+1, index_y+1) zxy(index_x+1, index_y) ...
zxy(index_x+1, index_y+1)];          %C 矩阵
fz = Hl*C*Hr;
```

例 4-19 分片双三次埃尔米特插值应用实例。根据下面的数据点（表格中的值是 z 值，按照 $z=\sin x\cos y$ 给出）计算当 $x=2.6, y=5.2$ 时 z 的值。

y \ x	1	2	3	4	5	6
2	−0.3502	−0.833	−0.55	0.2387	0.808	0.6344
3	−0.3784	−0.9002	−0.5944	0.2579	0.8731	0.6855
4	−0.0587	−0.1397	−0.0922	0.04	0.1355	0.1064
5	0.3149	0.7492	0.4947	−0.2147	−0.7267	−0.5706
6	0.3991	0.9493	0.6268	−0.272	−0.9207	−0.7229
7	0.1163	0.2766	0.1826	−0.0793	−0.2683	−0.2107

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> x=1:6;
>> y=2:7;
>> z=sin(x)'*cos(y);
>> zx=cos(x)'*cos(y)
>> zy=-sin(x)'*sin(y);
>> zxy=-cos(x)'*sin(y);
>> fz=DH(x,y,z,zx,zy,zxy,2.6,5.2)
 %计算 x=2.6,y=5.2 时的插值输出值 fz
fz =    0.2322
>> sin(2.6)'*cos(5.2)    %计算真实值
ans =    0.2415
```

从例题的结果可以看出，插值结果较好。

4.11 小结

本章介绍了各种插值的方法，主要是多项式或分段多项式的插值方法，因为它们最便于计算和使用。近几十年来，插值法仍在不断发展中，特别是利用样条函数的插值，在精密机械加工、计算机图形学等领域有着广泛的应用。

第 5 章 函数逼近

本章主要介绍函数逼近和曲线拟合的概念和算法。函数逼近的概念是按照一定的准则，用简单的连续函数（或分段函数）来逼近较为复杂的函数（或是一组离散点）。

函数逼近的目的在于更方便地计算函数值和进行函数运算（包括微积分）等。曲线拟合倾向于在一定准则下用一个简单的函数来模拟一组已知数据的函数关系，曲线拟合并不要求拟合曲线严格通过每个数据点，只要求在每个数据点上的残差的某种组合在一定意义上达到最小就可以了。

通过本章，读者不仅能掌握常见的函数逼近和曲线拟合算法，而且还能熟练使用 MATLAB 编程来实现这些算法。

5.1 切比雪夫逼近

当一个连续函数定义在区间[-1,1]上时，它可以展开成切比雪夫级数。即：

$$f(x)=\sum_{n=0}^{\infty} f_n T_n(x)$$

其中$T_n(x)$为n次切比雪夫多项式，具体表达式可通过递推得出：

$$T_0(x)=1,\ \ T_1(x)=x,\ \ T_{n+1}(x)=2xT_n(x)-T_{n-1}(x)$$

它们之间满足如下的正交关系：

$$\int_{-1}^{1}\frac{T_n(x)T_m(x)\mathrm{d}x}{\sqrt{1-x^2}}=\begin{cases}0 & n\neq m\\ \dfrac{\pi}{2} & n=m\neq 0\\ \pi, & n=m=0\end{cases}$$

在实际应用中，可根据所需的精度来截取有限项数。切比雪夫级数中的系数由下式决定：

$$f_0=\frac{1}{\pi}\int_{-1}^{1}\frac{f(x)}{\sqrt{1-x^2}}\mathrm{d}x$$

$$f_n=\frac{2}{\pi}\int_{-1}^{1}\frac{T_n(x)f(x)}{\sqrt{1-x^2}}\mathrm{d}x$$

在 MATLAB 中编程实现的切比雪夫逼近法函数为：Chebyshev

功能：用切比雪夫多项式逼近已知函数

调用格式：f = Chebyshev (y,k)或 f = Chebyshev (y,k,x0)

其中，y：　已知函数；

　　　k：　逼近已知函数所需项数；

x0:	逼近点的 x 坐标;
f:	求得的切比雪夫逼近多项式或是在 x0 处的逼近值。

在 MATLAB 中实现切比雪夫逼近的代码如下所示:

```
function f = Chebyshev(y,k,x0)
%用切比雪夫多项式逼近已知函数
%已知函数: y
%逼近已知函数所需项数: k
%逼近点的 x 坐标: x0
%求得的切比雪夫逼近多项式或是在 x0 处的逼近值: f
syms t;
T(1:k+1) = t;
T(1) = 1;
T(2) = t;
c(1:k+1) = 0.0;
c(1)=int(subs(y,findsym(sym(y)),sym('t'))*T(1)/sqrt(1-t^2),t,-1,1)/pi;
%常数项
c(2)=2*int(subs(y,findsym(sym(y)),sym('t'))*T(2)/sqrt(1-t^2),t,-1,1)/pi;
%一次项系数
f = c(1)+c(2)*t;
for i=3:k+1
    T(i) = 2*t*T(i-1)-T(i-2);          %切比雪夫多项式的递推公式
    c(i) =
2*int(subs(y,findsym(sym(y)),sym('t'))*T(i)/sqrt(1-t^2),t,-1,1)/2;
                                       %展开项系数
    f = f + c(i)*T(i);                 %展开后的多项式
    f = vpa(f,6);

    if(i==k+1)
        if(nargin == 3)
            f = subs(f,'t',x0);        %已知点的切比雪夫逼近值
        else
            f = vpa(f,6);
        end
    end
end
```

例 5-1 切比雪夫逼近应用实例。用切比雪夫公式(取 6 项)逼近函数 $\frac{1}{2-x}$,并求当 $x=0.5$ 时的函数值。

解: 在 MATLAB 命令窗口中输入以下命令:

```
>> f=Chebyshev('1/(2-x)',6)
f =.455803+.277012*t+.243094*t^2+.647768e-1*t*(2.*t^2-1.)+.173569e-1*t*
(2.*t*(2.*t^2-1.)-1.*t)+.501051e-2*t*(2.*t*(2.*t*(2.*t^2-1.)-1.*t)-2.*t^2+1.)
+.134256e-2*t*(2.*t*(2.*t*(2.*t*(2.*t^2-1.)-1.*t)-2.*t^2+1.)-2.*t*(2.*t^2-1.)
+t)
>> f=Chebyshev('1/(2-x)',6,0.5)
```

```
f = 0.6293
```

事实上函数的准确值为$1/(2-0.5)=0.6667$。

5.2 勒让德逼近

勒让德逼近也要求被逼近函数定义在区间[−1,1]上，勒让德多项式也可通过递推来定义：

$$P_0(x)=1, P_1(x)=x$$
$$(n+1)P_{n+1}(x)=(2n+1)xP_n(x)-nP_{n-1}(x)$$

它们之间满足如下的正交关系：

$$\int_{-1}^{1}P_n(x)P_m(x)\mathrm{d}x=\begin{cases}0 & n\neq m\\ \dfrac{1}{2n+1} & n=m\end{cases}$$

勒让德级数中的系数由下式决定：

$$f_n=\frac{2n+1}{2}\int_{-1}^{1}P_n(x)f(x)\mathrm{d}x$$

在 MATLAB 中编程实现的勒让德逼近法函数为：Legendre

功能：用勒让德多项式逼近已知函数

调用格式：f = Legendre (y,k)或 f = Legendre (y,k,x0)

其中，y：　已知函数；

k：　逼近已知函数所需项数；

x0：　逼近点的 x 坐标；

f：　求得的勒让德逼近多项式或是在 x0 处的逼近值。

在 MATLAB 中实现勒让德逼近的代码如下所示：

```
function f = Legendre(y,k,x0)
%用勒让德多项式逼近已知函数
%已知函数：y
%逼近已知函数所需项数：k
%逼近点的 x 坐标：x0
%求得的勒让德逼近多项式或是在 x0 处的逼近值：f
syms t;
P(1:k+1) = t;
P(1) = 1;
P(2) = t;
c(1:k+1) = 0.0;
c(1)=int(subs(y,findsym(sym(y)),sym('t'))*P(1),t,-1,1)/2; %常数项
c(2)=int(subs(y,findsym(sym(y)),sym('t'))*P(2),t,-1,1)/2; %一次项系数
f = c(1)+c(2)*t;
for i=3:k+1
    P(i) = ((2*i-3)*P(i-1)*t-(i-2)*P(i-2))/(i-1);
    c(i) = int(subs(y,findsym(sym(y)),t)*P(i),t,-1,1)/2; %展开项系数
```

```
    f = f + c(i)*P(i);      %展开后的多项式
    if(i==k+1)
        if(nargin == 3)
            f = subs(f,'t',x0);     %已知点的勒让德逼近值
        else
            f = vpa(f,6);
        end
    end
end
```

例 5-2 勒让德逼近应用实例。用勒让德公式（取 6 项）逼近函数 $\frac{1}{2-x}$，并求当 $x=0.5$ 时的函数值。

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> f=Legendre('1/(2-x)',6)
f =.539128+.955158e-1*t+.305351e-1*t^2+.154825e-2*(7.50000*t^2-2.50000)
*t +.275682e-3*(2.33333*(7.50000*t^2-2.50000)*t-4.66667*t)*t+.565953e-4*
(2.25000*(2.33333*(7.50000*t^2-2.50000)*t-4.66667*t)*t-10.1250*t^2+3.37500)
*t+.116697e-4*(2.20000*(2.25000*(2.33333*(7.50000*t^2-2.50000)*t-4.66667*t)
*t-10.1250*t^2+3.37500)*t-2.93333*(7.50000*t^2-2.50000)*t+5.86667*t)*t
>> f=Legendre('1/(2-x)',6,0.5)
f = 0.5935
```

从逼近结果看，函数的准确值为 $1/(2-0.5)=0.6667$，与上例相比可以看出勒让德级数的收敛速度比切比雪夫级数慢（因为取 6 项逼近，但是精度还是很差，也就是说为了达到切比雪夫级数相同的精度，勒让德级数需要取更多的项数）。

5.3 帕德逼近

帕德（Pade）逼近是一种有理分式逼近。逼近公式如下所示：

$$f(x)\approx\frac{\sum_{k=0}^{L}p_kx^k}{1+\sum_{k=1}^{M}q_kx^k}$$

大量实验表明，当 $L+M$ 为常数时，取 $L=M$，帕德逼近的精确度最好，而且速度最快。此时分子与分母中的系数可通过以下方法求解。

首先，求解线性方程组 $Aq=b$，得到 $(q_1,q_2,\cdots,q_n)$ 的值。其中

$$A=\begin{bmatrix}a_1 & a_2 & \cdots & a_n\\ a_2 & a_3 & \cdots & a_{n+1}\\ \vdots & \vdots & & \vdots\\ a_n & a_{n+1} & \cdots & a_{2n-1}\end{bmatrix},q=\begin{bmatrix}q_n\\ q_{n-1}\\ \vdots\\ q_1\end{bmatrix},b=\begin{bmatrix}-a_{n+1}\\ -a_{n+2}\\ \vdots\\ -a_{2n}\end{bmatrix}$$

$$a_0=f(0),a_n=\frac{1}{n!}\frac{d^nf(0)}{dx^n}$$

然后通过下式求出 $(p_0, p_1, \cdots, p_n)$ 的值。

$$p_0 = a_0, q_0 = 1, p_n = \sum_{i=0}^{n} q_i a_{n-i}$$



函数的帕德逼近不一定存在。

在 MATLAB 中编程实现的帕德逼近法函数为：Pade

功能：用帕德形式的有理分式逼近已知函数

调用格式：f = Pade (y,n)或 f = Pade (y,n,x0)

其中，y：　　已知函数；

n：　　帕德有理分式的分母多项式的最高次数；

x0：　　逼近点的 x 坐标；

f：　　求得的帕德有理分式或是在 x0 处的逼近值。

在 MATLAB 中实现函数的帕德逼近的代码如下所示：

```
function f = Pade(y,n,x0)
%用帕德形式的有理分式逼近已知函数
%已知函数：y
%帕德有理分式的分母多项式的最高次数：n
%逼近点的 x 坐标：x0
%求得的帕德有理分式或是在 x0 处的逼近值：f
syms t;
A = zeros(n,n);
q = zeros(n,1);
p = zeros(n+1,1);
b = zeros(n,1);
yy = 0;
a(1:2*n) = 0.0;
for(i=1:2*n)
    yy = diff(sym(y),findsym(sym(y)),n);
    a(i) = subs(sym(yy), findsym(sym(yy)), 0.0)/factorial(i);
end;
for(i=1:n)
    for(j=1:n)
        A(i,j)=a(i+j-1);
    end;
    b(i,1) = -a(n+i);
end;
q = A\b;            %求出分母的多项式系数
p(1) = subs(sym(y),findsym(sym(y)),0.0);
for(i=1:n)          %求出分子的多项式系数
    p(i+1) = a(n)+q(i)*subs(sym(y),findsym(sym(y)),0.0);
    for(j=2:i-1)
        p(i+1)=p(i+1)+q(j)*a(i-j);
    end
end
```

```
f_1 = 0;
f_2 = 1;
for(i=1:n+1)
    f_1 = f_1 + p(i)*(t^(i-1));
end
for(i=1:n)
    f_2 = f_2 + q(i)*(t^i);
end
if(nargin == 3)
    f = f_1/f_2;                %帕德逼近分式
    f = subs(f,'t',x0);         %已知点的帕德逼近值
else
    f = f_1/f_2;
    f = vpa(f,6);
end
```

例 5-3 帕德逼近应用实例。用帕德公式（取 4 项）逼近函数$\dfrac{1}{1-x}$，并求当 $x=0.5$ 时的函数值。

解：在 MATLAB 命令窗口中输入以下命令：

```
>> f=Pade('1/(1-x)',4)
f =(1.+1.00060*t+.988095*t^2+.821429*t^3+2.92857*t^4)/ (1.+.595238e
-3*t-.119048e-1*t^2+.107143*t^3-.500000*t^4)
>> f=Pade('1/(1-x)',4,0.5)
f = 2.0757
```

从逼近结果看，函数的准确值为$1/(1-0.5)=2$，而用 4 次有理分式的帕德逼近已经达到很高的精度了。

5.4 最佳一致多项式逼近

设函数$f(x)$在区间$[a,b]$的最佳一致逼近多项式为$p(x)=a_0+a_1x+\cdots+a_nx^n$，则确定$p(x)$系数的列梅兹算法步骤如下。

❶ 在区间$[t_0,t_1]$上取$n+1$次切比雪夫多项式的交错点组：

$$x_k=\frac{1}{2}[b+a+(b-a)\cos\frac{(n-k+1)\pi}{n+1}]\quad k=0,1,\cdots,n+1$$

作为初始点集。

❷ 将点集$\{x_0,x_1,\cdots,x_n,x_{n+1}\}$代入下式：

$$f(x_k)-\sum_{j=0}^{n}a_jx_k^j=(-1)^k\mu\quad (k=0,1,\cdots,n+1)$$

得到以$a_0,a_1,\cdots,a_n,\mu$为未知数的线性方程组，求解此方程组得到初始的逼近多项式$p(x)$。

❸ 求使$|f(x)-p(x)|$在$[t_0,t_1]$取最大值的x，设其为$\tilde{x}$，按下面的算法确定新的点集：

- 如果 $\tilde{x}$ 在 a 和 x_1 之间，并且 $f(x_1)-p(x_1)$ 和 $f(\tilde{x})-p(\tilde{x})$ 同号，则用 $\tilde{x}$ 代替 x_0，构成新的点集；
- 如果 $\tilde{x}$ 在 x_n 和 b 之间，并且 $f(x_n)-p(x_n)$ 和 $f(\tilde{x})-p(\tilde{x})$ 同号，则用 $\tilde{x}$ 代替 x_{n+1}，构成新的点集；
- 如果 $\tilde{x}$ 在某个 x_i 和 x_{i+1} 之间，并且 $f(x_i)-p(x_i)$ 和 $f(\tilde{x})-p(\tilde{x})$ 同号，则用 $\tilde{x}$ 代替 x_{i+1}，构成新的点集；

❹ 将❸中得到的新的点集代替旧的点集，求出新的 $a_0,a_1,\cdots,a_n,\mu$，如果新的 μ 与旧的 μ 的差在给定的精度范围内，则停止，否则重复上述过程。

在 MATLAB 中编程实现的列梅兹算法为：lmz

功能：用列梅兹算法确定函数的最佳一致逼近多项式

调用格式：[coff,err]=lmz(func,m,a,b,eps)

其中，func：　已知函数；

　　m：　最佳一致逼近多项式的最高次数；

　　a：　逼近区间的左端点；

　　b：　逼近区间的右端点；

　　eps：逼近误差的收敛控制精度；

　　coff：逼近多项式的系数；

　　err：逼近误差。

在 MATLAB 中实现列梅兹算法的代码如下所示：

```
function [coff,err]= lmz(func,m,a,b,eps)
%列梅兹算法确定函数的最佳一致逼近多项式
%已知函数：func
%最佳一致逼近多项式的最高次数：m
%逼近区间的左端点：a
%逼近区间的右端点：b
%逼近误差的收敛控制精度：eps:;
%逼近多项式的系数：coff
%逼近误差：err
if(nargin == 4)
   eps=1.0e-6;
end
syms v;
maxv = 0.0;
max_x = a;                    %记录abs(f(x)-p(x))取最大值的x
for k=0:m
  px(k+1)=power(v,k);
end                           %p(x)多项式
for  i=1:m+2
    x(i)=0.5*(a+b+(b-a)*cos(3.14159265*(m+2-i)/(m+1)));
    fx(i)=subs(sym(func), findsym(sym(func)),x(i));
```

```
end                          %初始的 x 和 f(x)
A = zeros(m+2,m+2);
for i=1:m+2
    for j=1:m+1
        A(i,j)=power(x(i),j-1);
    end
    A(i,m+2)=(-1)^i;
end
c =A\transpose(fx);            %p(x)的初始系数
u = c(m+2);                    %算法中的 u
tol = 1;                       %精度
while(tol>eps)
    t = a;
    while(t<b)           %此循环找出 abs(f(x)-p(x))取最大值的 x
        t = t + 0.05*(b-a)/m;
        px1 = subs(px,'v',t);
        pt = px1*c(1:m+1);
        ft = subs(sym(func), findsym(sym(func)),t);
        if abs(ft-pt)>maxv
            maxv = abs(ft-pt);
            max_x = t;
        end
    end
    if max_x>b
        max_x = b;
    end
    %以下可参考算法的三个确定新点集的情况
    if (a<=max_x)&&(max_x<=x(2))              %第一种情况
        f0 = subs(sym(func), findsym(sym(func)),x(2));
        px1 = subs(px,'v',x(2));
        pt = px1*c(1:m+1);
        d1 = f0 - pt;
        fm = subs(sym(func), findsym(sym(func)),max_x);
        pm1 = subs(px,'v',max_x);
        pm = pm1*c(1:m+1);
        d2 = fm - pm;
        if d1*d2>0
            x(2) = max_x;
        end
    else
        if (x(m+1)<=max_x)&&(max_x<=b)     %第二种情况
            f0 = subs(sym(func), findsym(sym(func)),x(m+1));
            px1 = subs(px,'v',x(m+1));
            pt = px1*c(1:m+1);
            d1 = f0 - pt;
            fm = subs(sym(func), findsym(sym(func)),max_x);
            pm1 = subs(px,'v',max_x);
            pm = pm1*c(1:m+1);
```

```
            d2 = fm - pm;
            if d1*d2>0
                x(m+1) = max_x;
            end
        else                            %第三种情况
            for i=2:m
                if(x(i)<=max_x)&& (x(i+1)>=max_x)
                    index_x = i;
                    break;
                end
            end                         %找到 max_x 所在区间
            f0 = subs(sym(func), findsym(sym(func)),x(index_x));
            px1 = subs(px,'v',x(index_x));
            pt = px1*c(1:m+1);
            d1 = f0 - pt;
            fm = subs(sym(func), findsym(sym(func)),max_x);
            pm1 = subs(px,'v',max_x);
            pm = pm1*c(1:m+1);
            d2 = fm - pm;
            if d1*d2>0
                x(index_x) = max_x;
            end
        end
    end
    for  i=1:m+2                        %重新计算 f(x)
        fx(i)=subs(sym(func), findsym(sym(func)),x(i));
    end
    for i=1:m+2
        for j=1:m+1
            A(i,j)=power(x(i),j-1);
        end
        A(i,m+2)=(-1)^i;
    end
    c =A\transpose(fx);                 %重新计算 p(x)的系数
    tol = abs(c(m+2)-u);
    u = c(m+2);
end
coff = c(1:m+1);
err = u;
```

例 5-4 最佳一致多项式逼近应用实例。求函数 $y=\sin x$ 在区间[0,7]上的 5 次最佳一致多项式逼近。

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> syms v;
>> y=sin(v);
>> [coff,err]=lmz(y,5,0,7)
coff = -0.0202
        1.1026
```

```
        0.0051
       -0.2809
        0.0670
       -0.0043
err = -0.0202
```

从结果可以得到函数 $y=\sin x$ 在区间[0,7]上的 5 次最佳一致逼近多项式为：

$$y=-0.0202+1.1026x+0.0051x^2-0.2809x^3+0.0670x^4-0.0043x^5$$

逼近误差为-0.0202。

5.5 最佳平方多项式逼近

求定义在区间 $[t_0,t_1]$ 上的已知函数最佳平方逼近多项式的算法介绍如下。

❶ 设已知函数 $f(x)$ 的最佳平方逼近多项式为 $p(x)=a_0+a_1x+\cdots+a_nx^n$，由最佳平方逼近的定义有：

$$\frac{\partial F(a_0,a_1,\cdots,a_n)}{\partial a_i}=0 \quad (i=0,1,2,\cdots,n)$$

其中 $F(a_0,a_1,\cdots,a_n)=\int_{t_0}^{t_1}(f(x)-a_0-a_1x-\cdots-a_nx^n)^2\mathrm{d}x$

❷ 形成多项式 $p(x)$ 系数的求解方程组：

$$Ca=D$$

其中：

$$C=\begin{bmatrix} \int_a^b \mathrm{d}x & \int_a^b x\mathrm{d}x & \cdots & \int_a^b x^{n-1}\mathrm{d}x & \int_a^b x^n\mathrm{d}x \\ \int_a^b x\mathrm{d}x & \int_a^b x^2\mathrm{d}x & \cdots & \int_a^b x^n\mathrm{d}x & \int_a^b x^{n+1}\mathrm{d}x \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \int_a^b x^{n-1}\mathrm{d}x & \int_a^b x^n\mathrm{d}x & \cdots & \int_a^b x^{2n-2}\mathrm{d}x & \int_a^b x^{2n-1}\mathrm{d}x \\ \int_a^b x^n\mathrm{d}x & \int_a^b x^{n+1}\mathrm{d}x & \cdots & \int_a^b x^{2n-1}\mathrm{d}x & \int_a^b x^{2n}\mathrm{d}x \end{bmatrix}$$

$$D=\begin{bmatrix} \int_a^b f(x)\mathrm{d}x \\ \int_a^b f(x)x\mathrm{d}x \\ \vdots \\ \int_a^b f(x)x^{n-1}\mathrm{d}x \\ \int_a^b f(x)x^n\mathrm{d}x \end{bmatrix}$$

在 MATLAB 中编程实现的最佳平方多项式逼近函数为：ZJPF

功能：求已知函数的最佳平方逼近多项式

调用格式：coff=ZJPF(func,n,a,b)

其中，func：	已知函数；
m：	最佳平方逼近多项式的最高次数；
a：	逼近区间的左端点；
b：	逼近区间的右端点；
coff：	逼近多项式的系数。

在 MATLAB 中实现最佳平方多项式逼近函数的代码如下所示：

```
function coff=ZJPF(func,n,a,b)
%求已知函数的最佳平方逼近多项式
%已知函数：func
%最佳平方逼近多项式的最高次数：n
%逼近区间的左端点：a
%逼近区间的右端点：b
%逼近多项式的系数：coff
C = zeros(n+1,n+1);
var = findsym(sym(func));
func = func/var;
for i=1:n+1
    C(1,i)=(power(b,i)-power(a,i))/i;     %算法中的 C 矩阵的第一行
    func = func*var;
    d(i,1)=int(sym(func),var,a,b);        %算法中的 D 向量的第一行
end
for i=2:n+1
    C(i,1:n)=C(i-1,2:n+1);
    f1 = power(b,n+i);
    f2 = power(a,n+i);
    C(i,n+1)=(f1-f2)/(n+i);               %形成 C 矩阵
end
coff = C\d;                               %求解逼近多项式的系数
```

例 5-5 最佳平方多项式逼近应用实例。求函数 $y=\sin x$ 在区间 $[0,7]$ 上的 5 次最佳平方多项式逼近。

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> syms v;
>> y=sin(v);
>> coff=ZJPF(y,5,0,7)
coff =[   -.878e-2]
      [      .9902]
      [      .1322]
      [    -.33266]
      [  .75749e-1]
      [ -.47944e-2]
```

从结果可以得到函数 $y=\sin x$ 在区间 $[0,7]$ 上的 5 次最佳一致逼近多项式为：

$$y=-0.00878+0.9902x+0.1322x^2-0.33266x^3+0.075749x^4-0.0047944x^5$$

5.6 傅立叶逼近

当被逼近的函数为周期函数时，用代数多项式来逼近效率不高，而且误差也较大，这时用三角多项式来逼近是较好的选择。

三角多项式逼近也就是傅立叶逼近，任一周期函数都可以展开为傅立叶级数，通过选取有限的展开项数，就可以达到所需精度的逼近效果。

下面介绍连续周期函数和离散周期函数的傅立叶逼近的具体做法。

1. 连续周期函数的傅立叶逼近

对于连续周期函数，只要计算出其傅立叶展开系数即可。

在 MATLAB 中编程实现的连续周期函数的傅立叶逼近法函数为：FZZ

功能：用傅立叶级数逼近已知的连续周期函数

调用格式：[A0,A,B] = FZZ (func,T,n)

其中，func： 已知函数；

T： 已知函数的周期；

n： 展开级数的项数；

A0： 展开后的常数项；

A： 展开后的余弦项系数；

B： 展开后的正弦项系数。

在 MATLAB 中实现连续周期函数的傅立叶逼近的代码如下所示：

```
function [A0,A,B]=FZZ(func,T, n)
%用傅立叶级数逼近已知的连续周期函数
%已知函数：func
%已知函数的周期：T
%展开项数：n
%展开后的常数项：A0
%展开后的余弦项系数：A
%展开后的正弦项系数：B
syms t;
func = subs(sym(func), findsym(sym(func)),sym('t'));
A0=int(sym(func),t,0,T)/T;
for(k=1:n)
    A(k)=int(func*cos(2*pi*k*t/T), t,0,T)*2/T;
    A(k)=vpa(A(k),4);
    B(k)=int(func*sin(2*pi*k*t/T), t,0,T)*2/T;
    B(k)=vpa(B(k),4);
end
```

例 5-6 傅立叶逼近应用实例。用傅立叶级数（取 5 项）逼近函数 x，输出系数值。

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> [A0,A,B]=fzz('x',2*pi,5)
A0 =     pi
A =      [ 0., 0., 0., 0., 0.]
B =     [    -2.,    -1., -.6667, -.5000, -.4000]
```

由于函数 $y=x$ 是奇函数，因此展开后的余弦项系数都为 0。

2. 离散周期数据的傅立叶逼近

对于离散周期函数的数据拟合，只要计算出其离散傅立叶展开系数即可。其展开公式为：

$$y=\sum_{k=0}^{n-1}c_i\mathrm{e}^{ikx}$$

其中

$$c_k=\frac{1}{N}\sum_{n=0}^{N-1}f_n\mathrm{e}^{-ikn\frac{2\pi}{N}}\ (k=0,1,\cdots,n-1)$$

在 MATLAB 中编程实现的离散周期数据点傅立叶逼近法函数为：DFF

功能：离散周期数据点的傅立叶逼近

调用格式：c = DFF (f,N)

其中，f：　已知离散数据点；

N：　离散数据点的个数；

c：　离散傅立叶逼近系数。

在 MATLAB 中实现离散周期函数的傅立叶逼近的代码如下所示：

```
function c = DFF(f,N)
%离散周期数据点的傅立叶逼近
%已知离散数据点：f
%离散数据点的个数：N
%逼近系数：c
c(1:N)=0;
for(m=1:N)
    for(n=1:N)
        c(m)=c(m)+f(n)*exp(-i*m*n*2*pi/N);
    end
    c(m)=c(m)/N;
end
```

例 5-7 离散傅立叶逼近应用实例。对下列数据点进行离散傅立叶变换。

N	1	2	3	4	5	6
y	0.8415	0.9093	0.1411	–0.7568	–0.9589	–0.2794

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> y=[0.8415  0.9093  0.1411  -0.7568  -0.9589  -0.2794];
```

```
>> c = DFT(y,6)
c =  Columns 1 through 3
   -0.0926 - 0.5003i  -0.0260 - 0.0194i  -0.0251 + 0.0000i
    Columns 4 through 6
   -0.0260 + 0.0194i  -0.0926 + 0.5003i  -0.0172 - 0.0000i
```

对于实数序列来说，其离散傅立叶变换的结果一般是复数序列。

5.7 自适应逼近

函数 $f(x)$ 在区间 $[a,b]$ 上的自适应逼近是指在给定的精度下，找到节点序列

$$x_i(i=0,1,2,\cdots,n) \quad x_0=a, \quad x_n=b$$

使得

$$\max_{x_i \leqslant x \leqslant x_{i+1}} \left|p(x)-f(x)\right| \leqslant \varepsilon \quad i=0,1,2,\cdots,n$$

其中，ε 为给定精度。$p(x)$ 为以 x_i 为端点的分段函数。

根据分段函数 $p(x)$ 的形式，可分为自适应分段线性逼近和自适应样条逼近等。

5.7.1 自适应分段线性逼近

自适应分段线性逼近的 $p(x)$ 的分段表达式为：

$$p_i(x)=f(x_i)+\frac{f(x_{i+1})-f(x_i)}{x_{i+1}-x_i}(x-x_i), \quad x\in(x_i,x_{i+1})$$

在 MATLAB 中编程实现的自适应分段线性逼近函数为：SmartBJ

功能：用自适应分段线性法逼近已知函数

调用格式：[node,err] = SmartBJ(func,a,b,maxtol)

其中，func：　已知函数；

a：　逼近区间的左端点；

b：　逼近区间的右端点；

maxtol：　分段线性逼近允许的最大误差；

node：　分段线性逼近的区间节点；

err：　分段线性逼近实际的最大误差。

在 MATLAB 中实现自适应分段线性法逼近已知函数的代码如下所示：

```
function [node,err] = SmartBJ(func,a,b,maxtol)
%自适应分段线性法逼近已知函数
%已知函数：func
%逼近区间的左端点：a
%逼近区间的右端点：b
%分段线性逼近允许的最大误差：maxtol
%分段线性逼近的区间节点：node
%分段线性逼近实际的最大误差：err
```

```
function [node,err] = SmartBJ(func,a,b,maxtol)
format long;
node(1) = a;
node(2) = b;
num = 2;
if(b-a)<10
   n = 5;
else
   n = 10;
end
err = 0;
bSign = 1;
while (bSign)
   bSign = 0;
   knode = node;
   tnum = num;
   insert_num = 0;
   for i=1:(tnum-1)
      [mx,mv] = FindMX(func,knode(i),knode(i+1),n);
      %找到区间[knode(i),knode(i+1)]上的误差最大的点
      if mv > maxtol
      %如果误差超过给定精度，在此点将区间[knode(i),knode(i+1)]分为两段
         d(1:(i+insert_num)) = node(1:(i+insert_num));
         d(i+insert_num+1) = mx;
         num = num+1;
         d((i+insert_num+2):num) = node((i+insert_num+1):(num-1));
         node = d;
         bSign = 1;
         insert_num = insert_num + 1;
      else
         if(mv>err)
            err = mv;          %记录所有分段线性插值区间上的最大误差
         end
      end
   end
end
format short;
function [max_x,max_v] = FindMX(func,a,b,n)
format long;
eps = 1.e-3;
max_v = 0;
max_x = a;
fa = subs(sym(func), findsym(sym(func)),a);     %左端点函数值
fb = subs(sym(func), findsym(sym(func)),b);     %右端点函数值
step = n/5;
tol = 1;
tmp = 0;
while tol>eps
    t = a;
    for j=0:(n/step)        %此循环找出取最大值的 x
       t = a + j*step*(b-a)/n;
       pt = fa + (t-a)*(fb-fa)/(b-a);  %线性插值得出的函数值
```

```
            ft = subs(sym(func), findsym(sym(func)),t);
            if abs(ft-pt)>max_v                     %abs(f(x)-p(x))
              max_v = abs(ft-pt);                   %记录最大误差
              max_x = t;                            %记录此点坐标
            end
        end
        tol = abs(max_x-tmp);
        tmp = max_x;
        step = step/2;
    end
    format short;
```

例 5-8 自适应分段线性逼近应用实例。在区间[0,4]上分段线性逼近 $y=\sqrt{x}$，精度为 0.01。

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> syms v;
>> y=sqrt(v);
>> [node,err]=SmartBJ(y,0,4,0.01)
node =
  Columns 1 through 6
       0    0.0008    0.0039    0.0156    0.0344    0.0625
  Columns 7 through 12
    0.1375    0.2500    0.3850    0.5500    1.0000    1.5400
  Columns 13 through 15
    2.2000    3.0325    4.0000
err = 0.0096
```

从结果可以看出，为了达到给定精度 0.01，在区间[0,4]上用了 15 个节点，而且从节点横坐标的大小可以看出来，节点在 $x=0$ 附近比较密集，离 $x=4$ 越近则越稀疏，这是因为 $y=\sqrt{x}$ 在 $x=0$ 附近斜率比较大，曲线比较陡，变化比较剧烈，因此需要比较多的节点来逼近它，而到后面曲线越来越平缓，需要的节点也就相对少一些。

为了了解分段线性逼近的过程，可参考下面的三幅图。

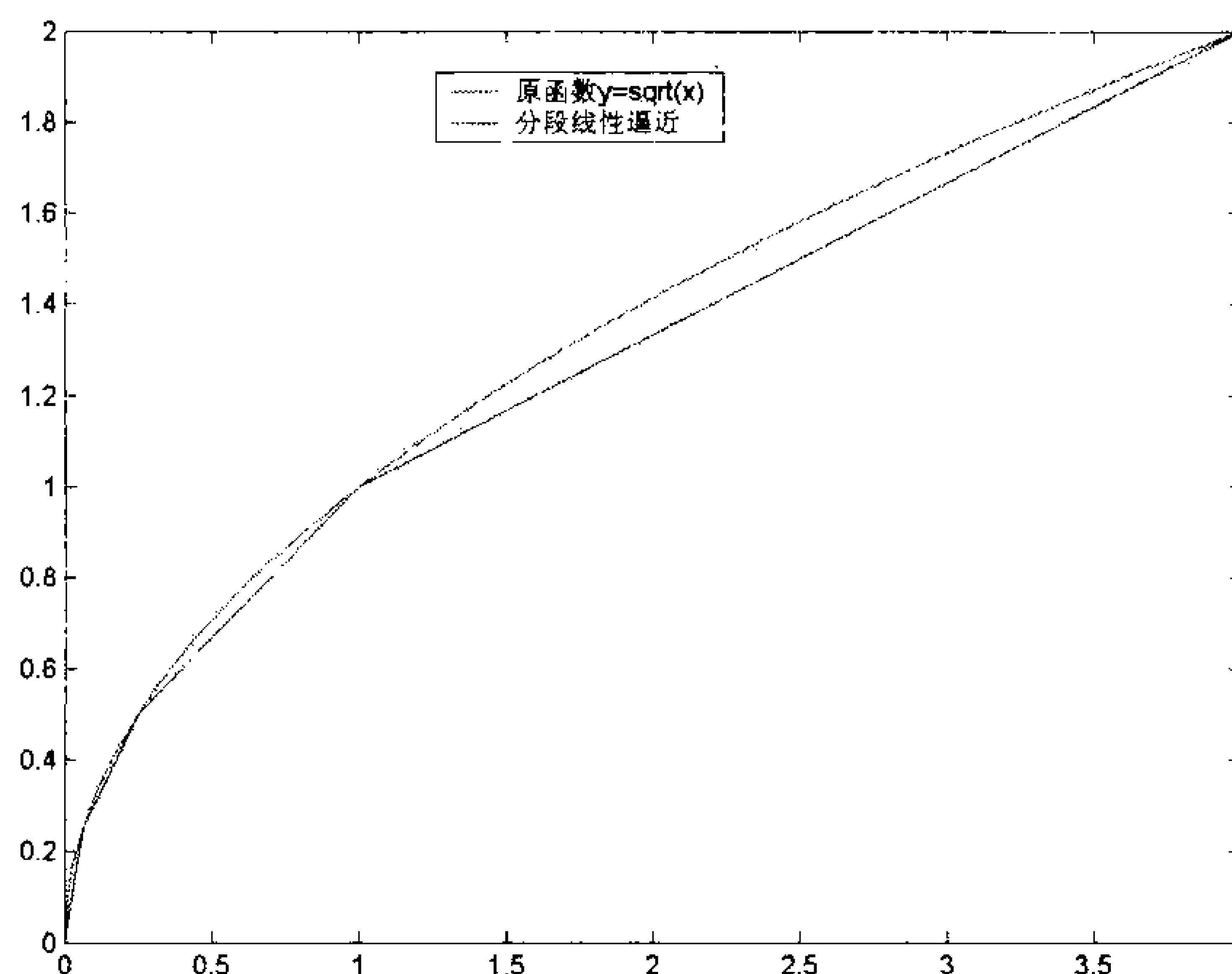


图 5-1 精度为 0.1 的自适应分段线性逼近

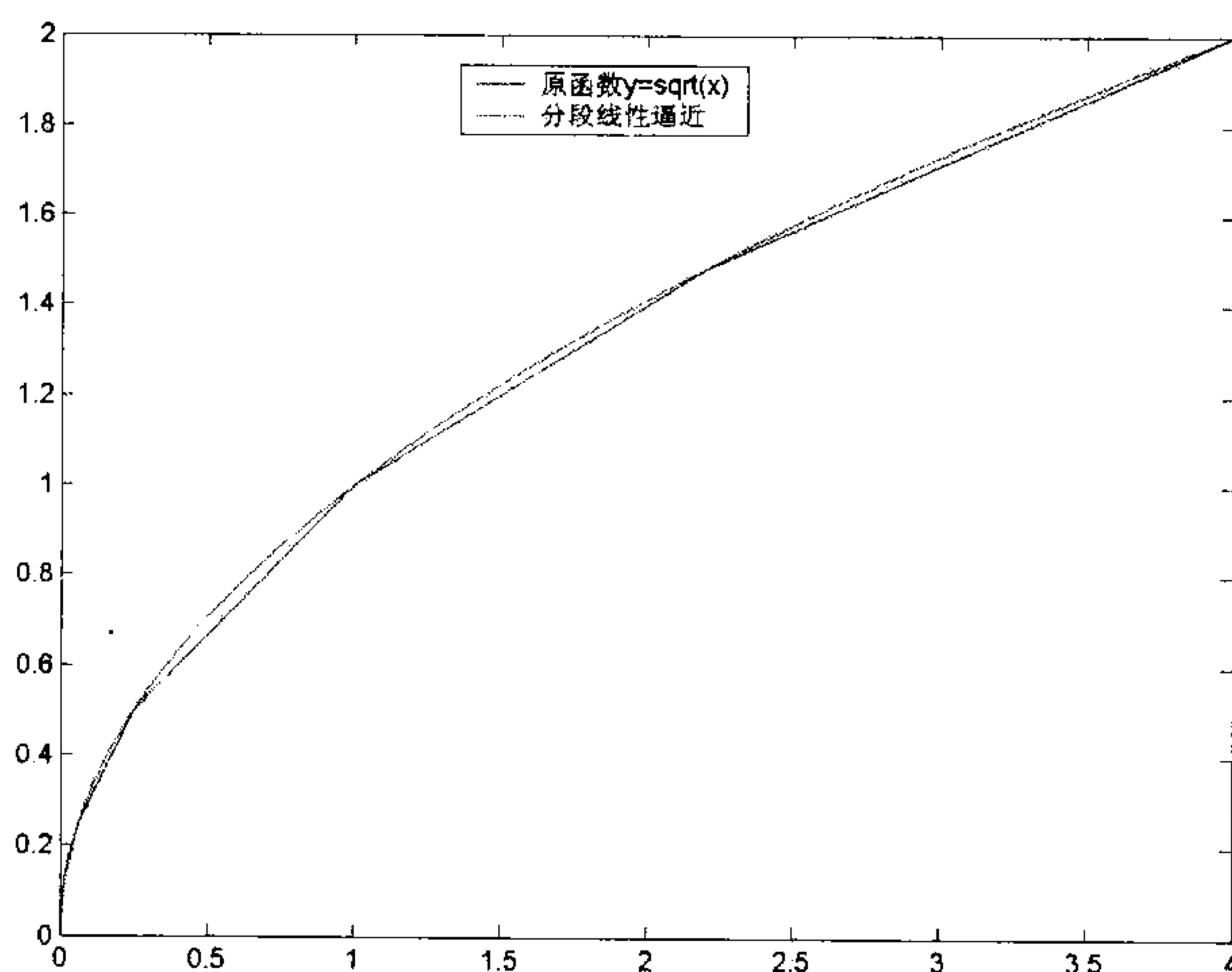


图 5-2　精度为 0.05 的自适应分段线性逼近

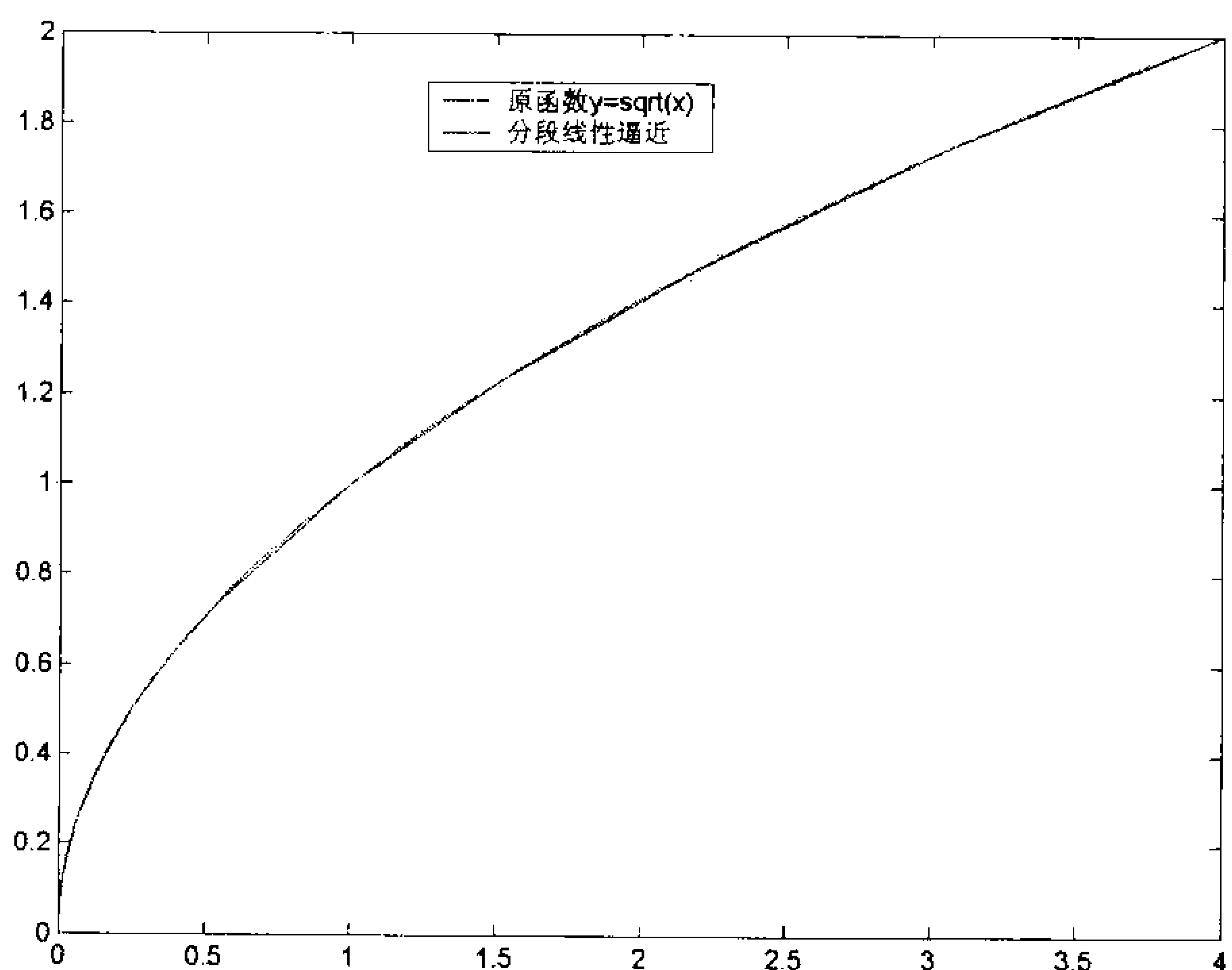


图 5-3　精度为 0.01 的自适应分段线性逼近

5.7.2　自适应样条逼近

自适应样条逼近（第一类）的 $p(x)$ 分段表达式为:

$$\begin{aligned} p_i(x) = & \frac{y_i}{h_i^3}[2(x-x_i)+h_i](x_{i+1}-x)^2 \\ & +\frac{y_{i+1}}{h_i^3}[2(x_{i+1}-x)+h_i](x-x_i)^2 \\ & +\frac{m_i}{h_i^2}(x-x_i)(x_{i+1}-x)^2-\frac{m_{i+1}}{h_i^2}(x_{i+1}-x)(x-x_i)^2 \\ & h_i = x_{i+1}-x_i (i=0,1,\cdots,N-1), x\in[x_i, x_{i+1}] \end{aligned}$$

各参数的意义可参照第 4.7 节。

在 MATLAB 中编程实现的自适应样条逼近（第一类）函数为：SmartBJ

功能：用自适应样条逼近（第一类）已知函数

调用格式：[node,err] = SmartBJ(func,a,b,maxtol)

其中，func：已知函数；

a：逼近区间的左端点；

b：逼近区间的右端点；

maxtol：分段样条逼近允许的最大误差；

node：分段样条逼近的区间节点；

err：分段样条逼近实际的最大误差。

在 MATLAB 中实现自适应样条逼近已知函数的代码如下所示：

```
function [node,err] = SmartBJ(func,a,b,maxtol)
%自适应样条逼近已知函数
%已知函数：func
%逼近区间的左端点：a
%逼近区间的右端点：b
%分段线性逼近允许的最大误差：maxtol
%分段线性逼近的区间节点：node
%分段线性逼近实际的最大误差：err
function [node,err] = SmartYTBJ(func,a,b,y_s,y_e,maxtol)
format long;
node(1) = a;
node(2) = b;
num = 2;
if(b-a)<10
    n = 5;
else
    n = 10;
end
err = 0;
bSign = 1;
while (bSign)
    bSign = 0;
    for l=1:num
        y(l) = subs(subs(sym(func), findsym(sym(func)),node(l)));
    end
    knode = node;
    tnum = num;
    insert_num = 0;
    for i=1:(tnum-1)
        pfx = ThrSample1(knode,y,y_s,y_e,((knode(i)+knode(i+1))/2));
        %上式给出每个分段区间上的样条插值函数
        [mx,mv] = FindMX(func,pfx,knode(i),knode(i+1),n);
        %找到区间[knode(i),knode(i+1)]上的误差最大的点
        if mv > maxtol
        %如果误差超过给定精度，在此点将区间[knode(i),knode(i+1)]分为两段，即将此点加
```

```
        %入节点数组中
            d(1:(i+insert_num)) = node(1:(i+insert_num));
            d(i+insert_num+1) = mx;
            num = num+1;
            d((i+insert_num+2):num) = node((i+insert_num+1):(num-1));
            node = d;
            bSign = 1;
            insert_num = insert_num + 1;
        else
            if(mv>err)
                err = mv;           %记录所有样条插值区间上的最大误差
            end
        end
    end
end
format short;
function [max_x,max_v] = FindMX(func,pfx,a,b,n)
format long;
eps = 1.e-3;
max_v = 0;
max_x = a;
fa = subs(sym(func), findsym(sym(func)),a);            %左端点函数值
fb = subs(sym(func), findsym(sym(func)),b);            %右端点函数值
step = n/5;
tol = 1;
tmp = 0;
while tol>eps
    t = a;
    for j=0:(n/step)                                    %此循环找出取最大值的 x
        t = a + j*step*(b-a)/n;
        pt = subs(sym(pfx), findsym(sym(pfx)),t);       %样条插值得出的函数值
        ft = subs(sym(func), findsym(sym(func)),t);
        if abs(ft-pt)>max_v                             %abs(f(x)-p(x))
           max_v = abs(ft-pt);                          %记录最大误差
           max_x = t;                                   %记录此点坐标
        end
    end
    tol = abs(max_x-tmp);
    tmp = max_x;
    step = step/2;
end
format short;
```

例 5-9 自适应样条逼近应用实例。在区间[1,4]上用自适应样条逼近 $y=\sqrt{x}$，精度分别为 0.1 和 0.01。

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> syms v;
>> y=sqrt(v);
```

```
>> [node,err]=SmartYTBJ(y,1,4,0.5,0.25,0.1)
node =     1     4
err =    0.0128
>> [node,err]=SmartYTBJ(y,1,4,0.5,0.25,0.01)
node =    1.0000    2.2000    4.0000
err =    0.0015
```

通过在程序代码的下面语句的地方设立断点:

```
pfx = ThrSample1(knode,y,y_s,y_e,((knode(i)+knode(i+1))/2))
```

可以查看到 $y=\sqrt{x}$ 在区间[1,4]上精度为 0.1 的样条逼近函数为

$$p(x)=\frac{1}{27}(2x+1)(x-4)^2+\frac{1}{27}(22-4x)(x-1)^2$$
$$+\frac{1}{9}(\frac{1}{2}x-\frac{1}{2})(4-x)^2-\frac{1}{9}(1-\frac{1}{4}x)(x-1)^2$$

化简得到: $p(x)=\frac{1}{108}x^3-\frac{1}{9}x^2+\frac{25}{36}x+\frac{11}{27}$

将 $y=\sqrt{x}$ 和 $p(x)=\frac{1}{108}x^3-\frac{1}{9}x^2+\frac{25}{36}x+\frac{11}{27}$ 在区间[1,4]的曲线绘制如图 5-4 所示。

采用相同的方法，在精度为 0.001 时，绘制如图 5-5 所示的对比曲线。

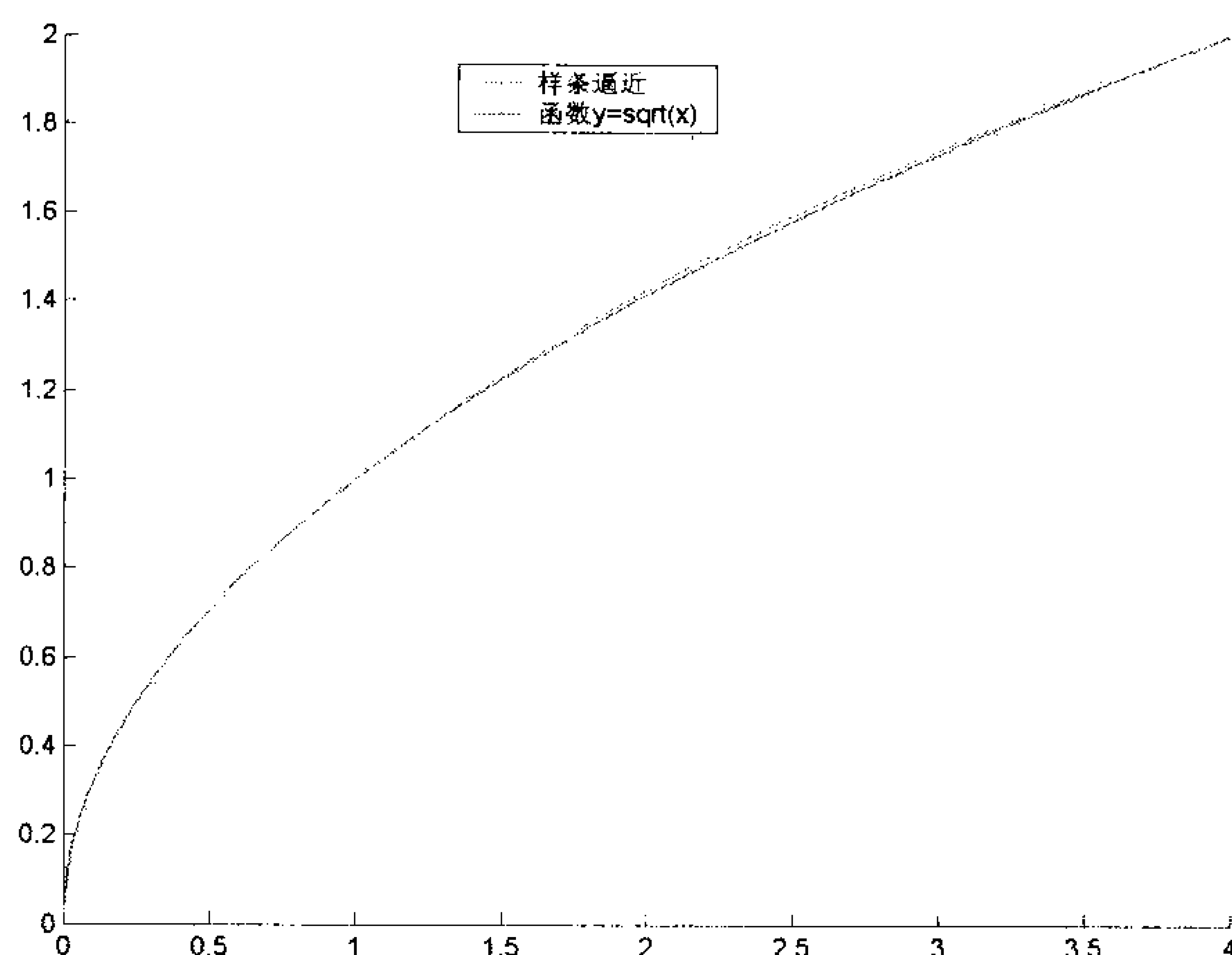


图 5-4 精度为 0.1 的自适应样条逼近

和上例比较，本例只用了 3 个节点就达到了 0.01 的逼近精度，可见样条函数的强大之处。

除了上面介绍的两种自适应逼近方法外，结合自适应方法和各种逼近理论可得到各种自适应逼近方法，如自适应切比雪夫逼近、自适应帕德逼近等。

自适应逼近是一种奇妙的逼近方法，它能在达到给定精度的前提下，采用尽可能少的逼近节点，而且节点的分布也十分合理，曲线变化剧烈的地方节点密，曲线平缓的地方节点稀。

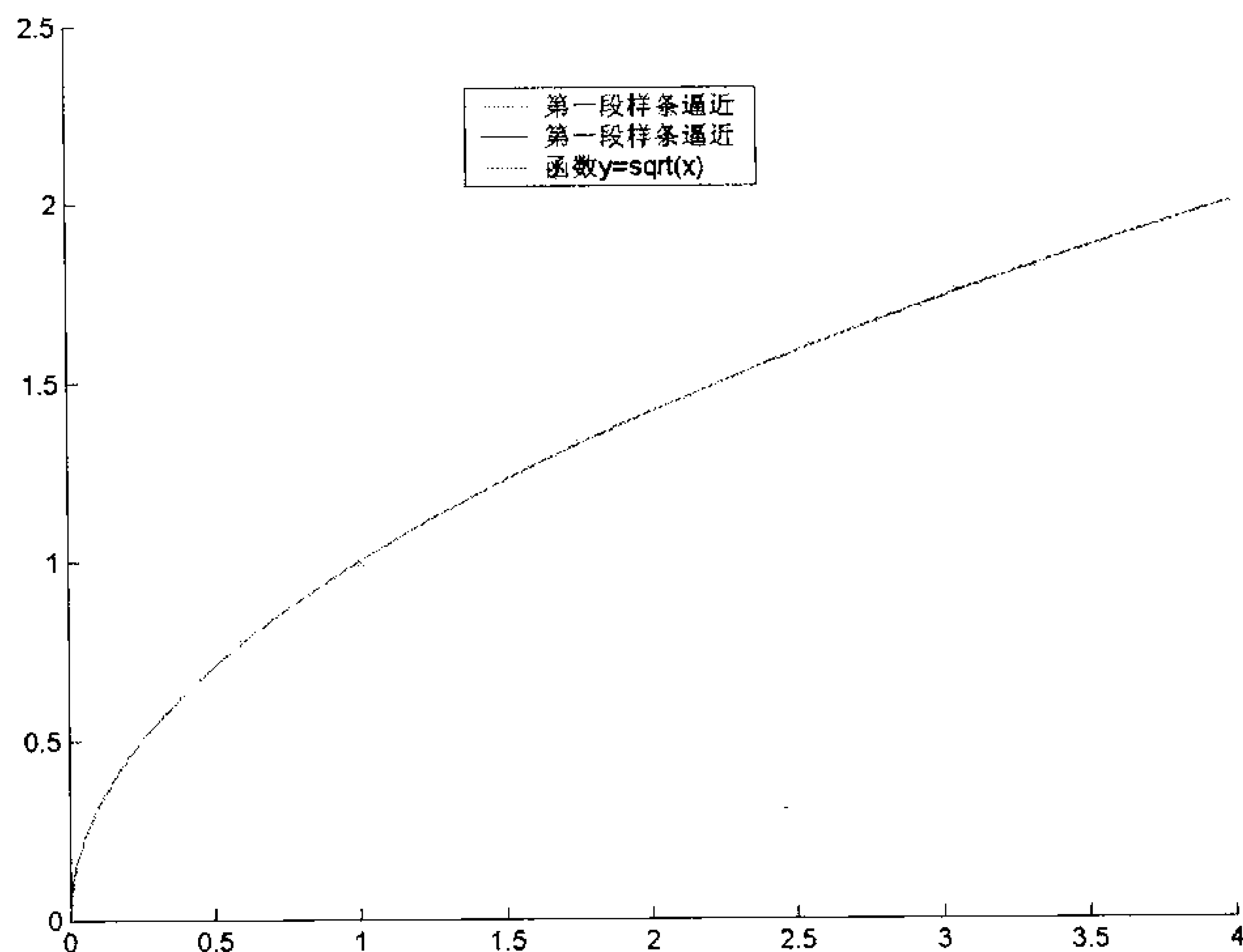


图 5-5　精度为 0.01 的自适应样条逼近

5.8 多项式曲线拟合

对给定的试验数据点 $(x_i, y_i)(i=1,2,\cdots,N)$，可构造 m 次多项式：

$$P(x)=a_0+a_1x+\cdots+a_mx^m \qquad (m<N)$$

由曲线拟合的定义，应该使得下式取极小：

$$\sum_{i=1}^{N}[\sum_{j=0}^{m}a_jx_i^j-y_i]^2$$

通过简单的运算可得出系数是下面线性方程组的解：

$$\begin{bmatrix} c_0 & c_1 & \cdots & c_m \\ c_1 & c_2 & \cdots & c_{m+1} \\ \vdots & \vdots & \cdots & \vdots \\ c_m & c_{m+1} & \cdots & c_{2m} \end{bmatrix}\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{bmatrix}=\begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_m \end{bmatrix}$$

其中

$$\begin{cases} c_k=\sum_{i=1}^{N}x_i^k,(k=0,1,\cdots,2m) \\ b_k=\sum_{i=1}^{N}y_ix_i^k,(k=0,1,\cdots,m) \end{cases}$$

在 MATLAB 中编程实现的多项式曲线拟合函数为：multifit

功能：离散试验数据点的多项式曲线拟合

调用格式：A = multifit (X,Y,m)

其中，X：　　试验数据点的 x 坐标向量；

Y：　　试验数据点的 y 坐标向量；

m：　　拟合多项式的次数；

A:	拟合多项式的系数向量。

在 MATLAB 中实现多项式曲线拟合的代码如下所示:

```
function A=multifit(X,Y,m)
%离散试验数据点的多项式曲线拟合
%试验数据点的 x 坐标向量: X
%试验数据点的 y 坐标向量: Y
%拟合多项式的次数: m
%拟合多项式的系数向量: A
N=length(X);
M=length(Y);
if(N ~= M)
    disp('数据点坐标不匹配! ');
    return;
end
c(1:(2*m+1))=0;
b(1:(m+1))=0;
for j=1:(2*m+1)            %求出 c 和 b
    for k=1:N
        c(j)=c(j)+X(k)^(j-1);
        if(j<(m+2))
            b(j)=b(j)+Y(k)*X(k)^(j-1);
        end
    end
end
C(1,:)=c(1:(m+1));
for s=2:(m+1)
    C(s,:)=c(s:(m+s));
end
A=b'\C;                    %直接求解法求出拟合系数
```

例 5-10 多项式曲线拟合应用实例。用二次多项式拟合下面的数据点。

x	1	2	3
y	2	5	10

解: 在 MATLAB 命令窗口中输入以下命令:

```
>> x = 1:3;
>> y = [1  5  10];
>> A = multifit(x,y,2)
A =    0.1282    0.3235    0.8718
```

即拟合多项式为: $p = 0.1282 + 0.3235x + 0.8718x^2$

5.9 线性最小二乘拟合

用线性函数: $y = ax + b$

拟合离散数据：　　　　　$(x_i, y_i)\ i = 0,1,2,\cdots,n$

在最小二乘意义上有：

$$F(a,b) = \sum_{i=0}^{n}(ax_i + b - y_i)^2 \quad \frac{\partial F}{\partial a} = 0,\ \frac{\partial F}{\partial b} = 0$$

解出 a 与 b 的值，则得到线性最小二乘函数。

在 MATLAB 中编程实现的线性最小二乘拟合函数为：LZXEC

功能：离散试验数据点的线性最小二乘拟合

调用格式：[a,b]=LZXEC(x,y)

其中，X：　　试验数据点的 x 坐标向量；

Y：　　试验数据点的 y 坐标向量；

a：　　拟合的一次项系数；

b：　　拟合的常数项。

在 MATLAB 中实现线性最小二乘拟合的代码如下所示：

```
function [a,b]=LZXEC(x,y)
%离散试验数据点的线性最小二乘拟合
%试验数据点的 x 坐标向量：X
%试验数据点的 y 坐标向量：Y
%拟合的一次项系数：a
%拟合的常数项：b
if(length(x) == length(y))
      n = length(x);
else
    disp('x 和 y 的维数不相等！');
    return;
end                      %维数检查
A = zeros(2,2);
A(2,2) = n;
B = zeros(2,1);
for i=1:n
    A(1,1) = A(1,1) + x(i)*x(i);
    A(1,2) = A(1,2) + x(i);
    B(1,1) = B(1,1) + x(i)*y(i);
    B(2,1) = B(2,1) + y(i);
end
A(2,1) = A(1,2);
s = A\B;
a = s(1);
b = s(2);
```

例 5-11　线性最小二乘拟合应用实例。用线性最小二乘拟合下面的数据。

x	1	2	3	4	5
y	1.5	1.8	4	3.4	5.7

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> x = 1:5;
>> y = [1.5  1.8  4 3.4  5.7];
>> [a,b] = LZXEC (x,y)
a =   1.0000
b =   0.2800
```

即拟合的式子为： $y = x + 0.28$

上面的程序还可以进一步完善，比如说加入计算相关系数的功能等。

5.10 正交多项式最小二乘拟合

正交多项式最小二乘拟合是选取一组在给定点上正交的多项式函数系 $\{B_i(x)\}$ 作为基函数进行最小二乘拟合。拟合的多项式记为：

$$p(x) = b_0 B_0(x) + b_1 B_1(x) + \cdots + b_m B_m(x)$$

其中

$$b_j = \frac{\sum_{i=0}^{n} y_i B_j(x_i)}{\sum_{i=0}^{n} B_j^2(x_i)}$$

基函数的构造公式如下所示：

$$B_0(x) = 1, B_1(x) = x - \alpha_0$$

$$B_{i+1}(x) = (x - \alpha_i) B_i(x) - \beta_i B_{i-1}(x)$$

$$\begin{cases} \alpha_i = \dfrac{\sum_{k=0}^{n} x_k B_i^2(x_k)}{\sum_{k=0}^{n} B_i^2(x_k)} \\ \beta_i = \dfrac{\sum_{k=0}^{n} B_i^2(x_k)}{\sum_{k=0}^{n} B_{i-1}^2(x_k)} \end{cases}$$

对给定的试验数据点 $(x_i, y_i)(i = 1, 2, \cdots, N)$，构造 m 次正交多项式最小二乘拟合的多项式的步骤介绍如下。

❶ 令 $B_0(x) = 1$，根据递推公式有：

$$b_0 = (\sum_{k=0}^{n} y_k)/(n+1), \alpha_0 = (\sum_{k=0}^{n} x_k)/(n+1)$$

则 $a_0 = b_0$。

❷ $B_1(x)=c_0+c_1x$，由递推公式有 $c_0=-\alpha_0, c_1=1$，$b_1=\dfrac{\sum\limits_{k=0}^{n} y_k B_1(x_k)}{\sum\limits_{k=0}^{n} B_1^2(x_k)}, \alpha_1=\dfrac{\sum\limits_{k=0}^{n} x_k B_1^2(x_k)}{\sum\limits_{k=0}^{n} B_1^2(x_k)}$，

$$\beta_1=\frac{\sum\limits_{k=0}^{n} B_1^2(x_k)}{\sum\limits_{k=0}^{n} B_0^2(x_k)}$$

更新逼近多项式的系数：$a_0=a_0+b_1c_0$，$a_1=b_1c_1$

❸ 对于 $t=2,3,\cdots,m$，设 $B_t(x)=r_0+r_1x+\cdots+r_tx^t$，$B_{t-1}(x)=s_0+s_1x+\cdots+s_{t-1}x^{t-1}$，$B_{t-2}(x)=w_0+w_1x+\cdots+w_{t-2}x^{t-2}$，由递推公式：

$$\begin{cases} r_t=s_{t-1} \\ r_{t-1}=-\alpha_{t-1}s_{t-1}+s_{t-2} \\ r_i=-\alpha_{t-1}s_i+s_{i-1}-\beta_{t-1}w_i \ (i=1,2,\cdots,t-2) \\ r_0=-\alpha_{t-1}s_0-\alpha_{t-1}w_0 \end{cases}$$

更新逼近多项式的系数：

$$\begin{cases} a_k=a_k+b_tr_k \quad k=0,1,2,\cdots,t-1 \\ a_t=b_tr_t \end{cases}$$

在 MATLAB 中编程实现的正交多项式最小二乘拟合函数为：ZJZXEC

功能：离散试验数据点的正交多项式最小二乘拟合

调用格式：a=ZJZXEC(x,y,m)

其中，x：　试验数据点的 x 坐标向量；

y：　试验数据点的 y 坐标向量；

m：　拟合多项式的次数；

a：　拟合多项式的系数向量。

在 MATLAB 中实现正交多项式最小二乘拟合的代码如下所示：

```
function a=ZJZXEC(x,y,m)
%离散试验数据点的正交多项式最小二乘拟合
%试验数据点的 x 坐标向量：x
%试验数据点的 y 坐标向量：y
%拟合多项式的次数：m
%拟合多项式的系数向量：a
if(length(x) == length(y))
    n = length(x);
else
    disp('x 和 y 的维数不相等！');
    return;
end                      %维数检查
syms v;
```

```
d = zeros(1,m+1);
q = zeros(1,m+1);
alpha = zeros(1,m+1);
for k=0:m
   px(k+1)=power(v,k);
end                          %x 的幂多项式
B2 = [1];
d(1) = n;
for l=1:n
    q(1) = q(1) + y(l);
    alpha(1) = alpha(1) + x(l);
end
q(1) = q(1)/d(1);
alpha(1) = alpha(1)/d(1);
a(1) = q(1);        %算法的第一步，求出拟合多项式的常数项
B1 = [-alpha(1) 1];
for l=1:n
    d(2) = d(2) + (x(l)-alpha(1))^2;
    q(2) = q(2) + y(l)*(x(l)-alpha(1));
    alpha(2) = alpha(2) + x(l)*(x(l)-alpha(1))^2;
end
q(2) = q(2)/d(2);
alpha(2) = alpha(2)/d(2);
a(1) = a(1)+q(2)*(-alpha(1));  %更新拟合多项式的常数项
a(2) = q(2);       %算法的第二步，求出拟合多项式的一次项系数
beta = d(2)/d(1);
for i=3:(m+1)
    B = zeros(1,i);
    B(i) = B1(i-1);
    B(i-1) = -alpha(i-1)*B1(i-1)+B1(i-2);
    for j=2:i-2
       B(j) = -alpha(i-1)*B1(j)+B1(j-1)-beta*B2(j);
    end
    B(1) = -alpha(i-1)*B1(1)-beta*B2(1);
    BF = B*transpose(px(1:i));
    for l=1:n
        Qx = subs(BF,'v',x(l));
        d(i) = d(i) + (Qx)^2;
        q(i) = q(i) + y(l)*Qx;
        alpha(i) = alpha(i) + x(l)*(Qx)^2;
    end
    alpha(i) = alpha(i)/d(i);
    q(i) = q(i)/d(i);
    beta = d(i)/d(i-1);
    for k=1:i-1
        a(k) = a(k)+q(i)*B(k);      %更新拟合多项式的系数
    end
    a(i) = q(i)*B(i);
    B2 = B1;
    B1 = B;
end
```

例 5-12 正交多项式最小二乘拟合应用实例。用四次正交多项式最小二乘拟合下面的数据。

x	1	2	3	4	5
y	1.5	1.8	4	3.4	5.7

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> x = 1:5;
>> y = [1.5  1.8  4  3.4  5.7];
>> a = ZJZXEC (x,y,4)
a =  18.2000  -32.8333  20.8167  -5.1167   0.4333
```

即拟合的多项式为：$y=18.2-32.8333x+20.8167x^2-5.1167x^3+0.4333x^4$

线性最小二乘和四次正交多项式最小二乘拟合的结果如图 5-6 所示：

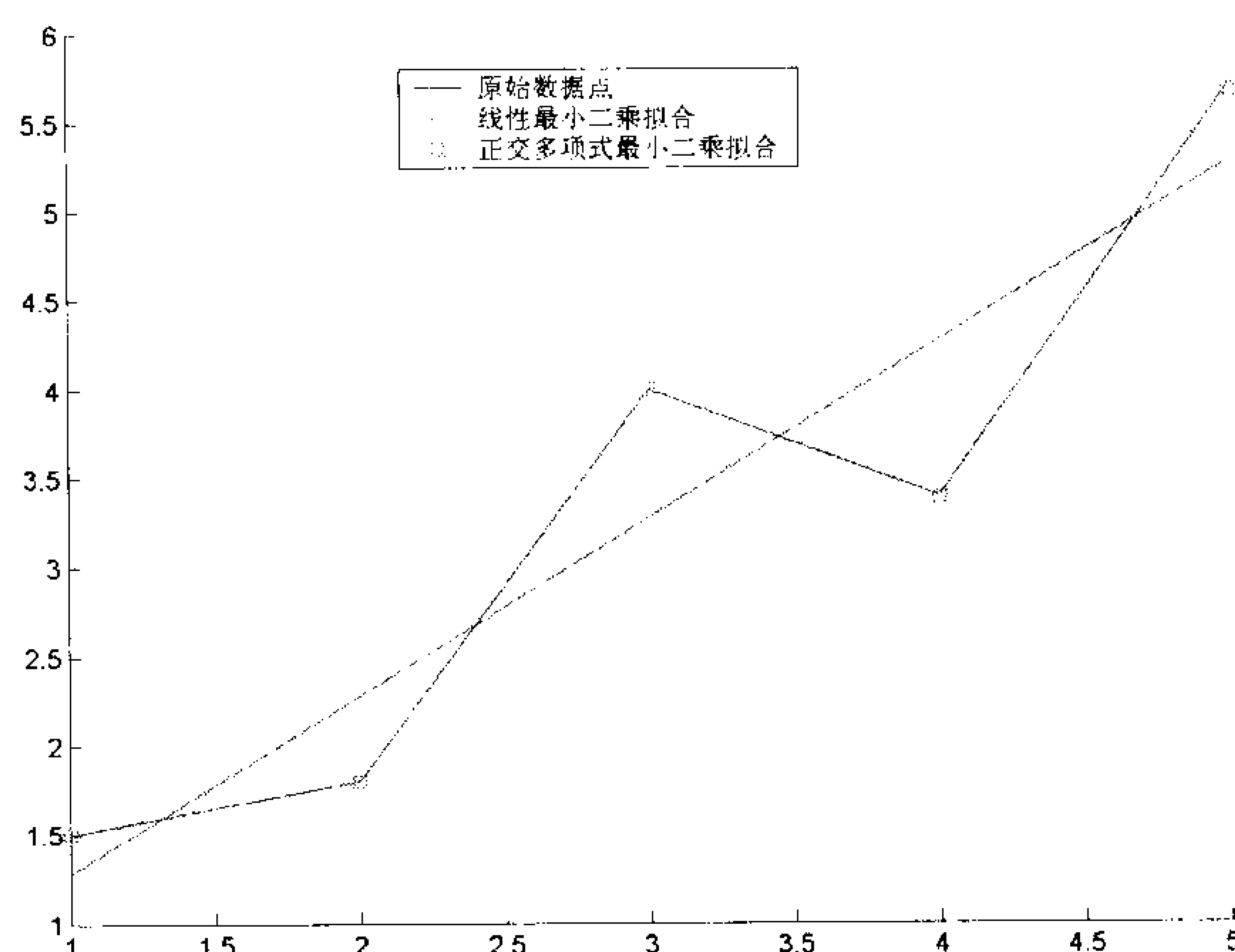


图 5-6　最小二乘拟合数据

5.11 小结

逼近是为了在计算机上更加方便地计算函数的值，因为多项式计算起来是最简单的，不管哪种逼近，它们都是使逼近函数和已知函数的误差在某种意义上达到最少，从而有最佳一致逼近、最佳平方逼近和最小二乘逼近。自适应逼近是适应性比较广的算法，但是它耗费时间比较长，相对来说，如果用多项式逼近的话，首选可用正交多项式最小二乘拟合方法。

第 6 章　矩阵特征值计算

物理、力学和工程技术中的很多问题在数学上都归结为求解矩阵特征值问题。当矩阵的阶数较小时，可以通过求特征多项式的根来得到特征值，即利用 MATLAB 中的函数来实现；当矩阵阶数比较大的时候，就需要用到一些数值方法，利用 MATLAB 编程也可方便实现。

通过本章，读者能够熟练掌握 MATLAB 中的特征值来计算相关函数，而且能通过编程实现多种矩阵特征值运算的算法。

6.1 特征值与特征向量

设 A 是 n 阶矩阵，x 是非零列向量，如果有数 λ 存在，满足 $Ax=\lambda x$，那么称 x 是矩阵 A 关于特征值 λ 的特征向量。

将 $Ax=\lambda x$ 改写为 $(A-\lambda I)x=0$，则称：

$$f(\lambda)=|A-\lambda I|=\lambda^n+a_1\lambda^{n-1}+\cdots+a_{n-1}\lambda+a_n$$

为矩阵 A 的特征多项式。

当 λ 是 A 的一个特征根时，它必然是 $f(\lambda)=0$ 的一个根。

6.2 条件数与病态矩阵

若矩阵 A 为非奇异，则称 $\|A\|\|A^{-1}\|$ 为矩阵 A 的条件数，其中 $\|\cdot\|$ 为 A 的某种范数。

由于矩阵范数的定义不同，因而其条件数也不同，但是由于矩阵范数的等价性，故在不同范数下的条件数也是等价的。

矩阵条件数的大小是衡量矩阵“坏”或“好”的标志。条件数大的矩阵称为病态矩阵或坏矩阵。一般来说，若矩阵的按模最大特征值与按模最小特征值之比值较大时，矩阵就会呈病态。特别地，当 A 的行列式值很小时，矩阵总是病态的。

设 X 为向量，常用的几种向量范数定义如下：

欧几里德范数，即 $\|X\|_2=\sqrt{\sum|x_k|^2}$；

∞-范数，即 $\|X\|=\max(abs(X))$；

1-范数，即 $\|X\|_1=\sum|x_k|$；

p-范数，$\|X\|_p=\sqrt[p]{\sum|x_k|^p}$。

对应于向量范数，矩阵 A 的范数定义如下：

欧几里德范数等于 A 的最大奇异值；

列范数等于 A 的列向量的 1-范数的最大值；

行范数等于 A 的行向量的 1-范数的最大值；

Frobenius 范数，即 $\|A\|_F=\sqrt{\sum\sum a_{ij}^2}$ 。

MATLAB 中提供了 norm 函数来求矩阵的范数，其具体用法如表 6-1 所示。

表 6-1 norm 函数

格　式	说　明
n = norm(A)	求 A 的欧几里德范数
n = norm(A,1)	求 A 的列范数
n = norm(A,2)	求 A 的欧几里德范数
n = norm(A,inf)	求 A 的行范数
n = norm(A, 'fro')	求 A 的 Frobenius 范数

不同的范数有相应不同的条件数，MATLAB 中提供了以下几个求矩阵条件数的函数。

- cond：求矩阵的条件数。

基本用法与功能介绍如下。

c = cond(X)：求 X 的 2-范数的条件数，即 X 的最大奇异值和最小奇异值的商。

c = cond(X,p)：求 p-范数的条件数，p 的值可以是 1、2、inf 或者'fro'。

- condest：求矩阵的条件数估计值。

基本用法与功能介绍如下。

c = condest (A)：求矩阵 A 的 1-范数的条件数的下界估值。

[c,v] = condest (A)：v 为向量，满足 $\|Av\|=\dfrac{\|A\|\cdot\|v\|}{c}$。

[c,v] = condest (A,t)：求上面的 c 和 v，同时显示出关于计算的步骤信息。如果 t=1，则计算的每步都显示出来；如果 t=–1，则给出商 c/rcond(A)。

- rcond：判断矩阵的病态程度。

基本用法与功能介绍如下。

c = rcond(A)：对于病态矩阵 A 来说，给出一个接近于 0 的数；对于非病态矩阵 A，则给出一个接近于 1 的数。

- condeig：求矩阵的特征值的条件数。

基本用法与功能介绍如下。

c = condeig(A)：返回短阵 A 的特征值的条件数。

[V, D, c] = condeig(A)：D 为 A 的特征值对角阵，V 为 A 的特征向量。

例 6-1 矩阵范数求取实例。求矩阵 A 的各种范数。

$$A=\begin{bmatrix}1 & 3 & 8\\ -5 & 2 & 9\\ 0 & 1 & 4\end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A= [1 3 8; -5 2 9; 0 1 4];
>> n1= norm(A)
>> n2= norm(A,1)              %列范数
>> n3= norm(A,inf)            %行范数
>> n4= norm(A,'fro')          %Frobenius 范数
```

输出计算结果为：

```
n1 =   13.5336
n2 =    21
n3 =    16
n4 =   14.1774
```

例 6-2 矩阵条件数求取实例。求矩阵 A 的各种条件数。

$$A=\begin{bmatrix}1 & 3 & 8\\ -5 & 2 & 9\\ 0 & 1 & 4\end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A= [1 3 8; -5 2 9; 0 1 4];
>> c1= cond(A)
>> c2= cond(A,1)              %1-范数的条件数
>> c3= cond(A,inf)            %行范数的条件数
>> c4= cond(A,'fro')          %Frobenius 范数的条件数
>> c5= rcond(A)               %判断矩阵的病态程度
```

输出计算结果为：

```
c1 =   40.5924
c2 =   85.1053
c3 =   61.4737
c4 =   42.6696
c5 =   0.0118
```

从 A 的条件数可以看出，A 是一个病态矩阵。

6.3 相似变换

相似矩阵有着相同的特征值，在 MATLAB 中，函数 balance 能够求出相似变换矩阵，它的基本用法与功能介绍如下。

$[T, B]$ = balance(A)：求相似变换矩阵 T 和平衡矩阵 B，满足 $B=T^{-1}AT$，且 T 为对角阵。

B = balance(A)：求平衡矩阵 B。

例 6-3　矩阵相似变换实例。利用函数 balance，计算矩阵$\begin{bmatrix} 1 & 3 & 0 \\ 3 & 9 & 4 \\ -8 & 6 & 2 \end{bmatrix}$的平衡矩阵和相似变换矩阵。

解： 在 MATLAB 命令窗口中输入：

```
>> J=[1 3 0;3 9 4;-8 6 2;];
>> [T,B]=balance(J)
```

输出计算结果为：

```
T =  0.5000       0         0
        0        1.0000   0
        0         0        2.0000
B =1.0000    6.0000        0
   1.5000    9.0000    8.0000
  -2.0000    3.0000    2.0000
>> B=balance(J)
B =1.0000    6.0000        0
   1.5000    9.0000    8.0000
  -2.0000    3.0000    2.0000
```

对于稀疏方阵（零元素较多），T 的形式可能会不同。

继续在 MATLAB 命令窗口中输入：

```
>> J=[1 2 0;2 4 0;8 3 9];
>> [T,B]=balance(J)
```

输出计算结果为：

```
T = 0     0     1
    0     1     0
    1     0     0
B = 9     3     8
    0     4     2
    0     2     1
```

上面的 T 矩阵为副对角线矩阵，是因为 J 的第三列有两个 0，在求 B 之前对 J 进行了行变换，如果想得到对角型的矩阵 T，可采用 noperm 选项。

继续在 MATLAB 命令窗口中输入：

```
>> [T,B]=balance(J,'noperm')
```

输出计算结果为：

```
T = 0.2500         0          0
         0    0.2500          0
         0         0     1.0000
B = 1.0000    2.0000          0
    2.0000    4.0000          0
    2.0000    0.7500     9.0000
```

这样得出的 T 矩阵就为对角阵了。如果矩阵 A 为对角阵，则得出的 T 矩阵为单位矩阵。

继续在 MATLAB 命令窗口中输入：

```
>> J=[1 2 3;2 4 6;3 6 9];
>> [T,B]=balance(J)
```

输出计算结果为：

```
T = 1    0    0
    0    1    0
    0    0    1
B = 1    2    3
    2    4    6
    3    6    9
```

求平衡矩阵的目的是调整矩阵各个元素之间的大小关系，降低其病态程度。

一般来说，改用平衡矩阵 B 来求特征值，会比直接用 A 来求特征值的精度要高。

6.4 特征值求取

特征值的求法有多种，有些方法只能求出一个特征值，如幂法；而有些方法能求出所有的特征值，如收缩法。有的只能求出实特征值，有的能求出复特征值，如特征多项式法。有的能附带求出特征向量，而有的只能求出特征值。下面由简单到复杂依次介绍各种求矩阵特征值的算法。

6.4.1 特征多项式法

此方法的思想十分简单，根据特征值的性质，它应该是特征多项式的根，因此可以通过求特征多项式的根来求特征值。

在 MATLAB 中编程实现的特征多项式法的函数为：Chapoly。

功能：通过求矩阵特征多项式的根来求其特征值。

调用格式：l = Chapoly (A)

其中，A 为已知矩阵；

l 为求得的矩阵特征值。

特征多项式法的 MATLAB 程序代码如下所示：

```
function l = Chapoly(A)
%特征多项式法求矩阵特征值
%已知矩阵：A
%求得的矩阵特征值：l
syms t;
N = size(A);
n = N(1,1);
y = det(A-t*eye(n,n));
l = solve(y);
```

```
l = vpa(l, 5);          %结果取五位精度
```

例 6-4 特征多项式求特征值应用实例。采用特征多项式法，求矩阵 A 的特征值。

$$A=\begin{bmatrix}11 & 5 & 8\\ -5 & 6 & 10\\ -2 & 3 & 4\end{bmatrix}$$

解： 在 MATLAB 命令窗口中输入：

```
>> A= [11 5 8; -5 6 10; -2 3 4];
>> l= Chapoly(A)
```

输出计算结果为：

```
l =   -.5721
     10.786-6.4035*i
     10.786+6.4035*i
```

由计算结果可知，矩阵 A 的特征值为–0.5721、10.786–6.4035*i 和 10.786+6.4035*i。

在实际使用时，会发现用特征多项式法求矩阵的特征值速度很慢，因此几乎不使用这种方法。

6.4.2 幂法

幂法是用来求矩阵的主特征值和主特征向量的迭代法。主特征值指的是按模最大的特征值，它对应的特征向量称为主特征向量。幂法的迭代过程为：

给定迭代初始值 $x^0 \neq 0$ 和误差限 $\varepsilon > 0$，

$$y^{k+1} = Ax^k$$

$$m^{k+1} = \max(y^{k+1})$$

$$x^{k+1} = \frac{y^{k+1}}{m^{k+1}}$$

如果 $\left|(m^{k+1} - m^k)\right| < \varepsilon$，结束。

其中，$\max(y^{k+1})$ 表示向量 y^{k+1} 按模最大的分量。幂法有如下的收敛性质：

$$\lim_{k\to\infty} x^k = {a_1}\big/{\max(a_1)},\quad \lim_{k\to\infty} m^k = \lambda_1$$

其中 λ_1 为主特征值，a_1 为对应的主特征向量。

> 在 MATLAB 中编程实现的幂法的函数为：pmethod。
>
> 功能：幂法求矩阵的主特征值及主特征向量。
>
> 调用格式：[l, v, s]=pmethod (A,x0,eps)
>
> 其中，A 为已知矩阵；
>
> x0 为迭代初始向量；
>
> eps 为迭代的精度；
>
> l 为求得的矩阵主特征值；

v 为求得的矩阵主特征向量；
s 为迭代步数。

幂法的 MATLAB 程序代码如下所示：

```
function [l,v,s] =pmethod(A,x0,eps)
%幂法求矩阵的主特征值及主特征向量
%已知矩阵：A
%迭代初始向量：x0
%迭代的精度：eps
%求得的矩阵特征值：l
%求得的矩阵主特征向量：v
%迭代步数：s
if nargin==2
   eps = 1.0e-6;
end
v = x0;                  %v 为主特征向量
M = 5000;                %迭代步数限制
m = 0;
l = 0;
for(k=1:M)
   y = A*v;
   m = max(y);           %m 为按模最大的分量
   v = y/m;
   if(abs(m - l)<eps)
      l = m;             %到所需精度，退出，l 为主特征值
      s = k;             %s 为迭代步数
      return;
   else
      if(k==M)
         disp('迭代步数太多，收敛速度太慢！');
         l = m;
         s = M;
      else
         l = m;
      end
   end
end
```



- 当矩阵为病态矩阵时，幂法的收敛速度非常慢；
- 初始迭代向量不能取零向量。

例 6-5 幂法求特征值应用实例。采用幂法求矩阵 A 的主特征值和主特征向量。

$$A=\begin{bmatrix}1 & 5 & 2\\ 6 & -1 & 7\\ 1 & 3 & 1\end{bmatrix}$$

解： 在 MATLAB 命令窗口中输入：

```
>> A  = [1 5 2; 6 -1 7; 1 3 1];
>> x0 = [ 1 1 1]';
>> [l,v,s]= pmethod(A,x0)
```

输出计算结果为：

```
l = 8.0407
v = 0.8661
    1.0000
    0.5491
s = 84
```

由计算结果可知，经过 84 步迭代，求出了矩阵 A 的主特征值为 8.0407，对应的特征向量为$[0.8661，1.0000，0.5491]^T$。

6.4.3　瑞利商加速幂法

如果矩阵为对称矩阵，可以用瑞利法来加快幂法的收敛速度。对于非零向量 x，它的瑞利商定义如下：

$$R(x)=\frac{(Ax,x)}{(x,x)}$$

瑞利商加速幂法的迭代过程如下：

给定迭代初始值 $x^0 \neq 0$ 和误差限$\varepsilon>0$，

$$y^{k+1}=Ax^k$$

$$m^{k+1}=\frac{(Ax^k,x^k)}{(x^k,x^k)}$$

$$x^{k+1}=\frac{y^{k+1}}{m^{k+1}}$$

如果$\left|(m^{k+1}-m^k)\right|<\varepsilon$,则结束。

当然，如果矩阵不对称，也可以用瑞利商加速幂法来求其特征值，但是加速的效果可能不是很明显。

在 MATLAB 中编程实现的瑞利商加速幂法的函数为：rpmethod。

功能：瑞利商加速幂法求对称矩阵的主特征值及主特征向量。

调用格式：[l, v, s]=rpmethod (A, x0, eps)。

其中，A 为已知矩阵；

x0 为迭代初始向量；

eps 为迭代的精度；

l 为求得的矩阵主特征值；

v 为求得的矩阵主特征向量；

s 为迭代步数。

瑞利商加速幂法的 MATLAB 程序代码如下所示：

```
function [l,v,s]=rpmethod(A,x0,eps)
%瑞利商加速幂法求对称矩阵的主特征值及主特征向量
%已知矩阵：A
%迭代初始向量：x0
%迭代的精度：eps
%求得的矩阵特征值：l
%求得的矩阵主特征向量：v
%迭代步数：s
if nargin==2
    eps = 1.0e-6;
end
v = x0;                          %v 为主特征向量
M = 500;                         %迭代步数限制
m = 0;
l = 0;
for(k=1:M)
    y = A*v;
    m = (y'*v)/(v'*v);           %m 为瑞利商
    if(rank(m)>1)
        m=(y*v')/(v*v');
    end
    v = y/m;
     if(abs(m - l)<eps)
        l = m;                   %到所需精度，退出，l 为主特征值
        s = k;                   %s 为迭代步数
        return;
    else
        if(k==M)
            disp('迭代步数太多，收敛速度太慢！');
             l = m;
             s = M;
        else
            l = m;
        end
    end
end
```

例 6-6 瑞利商加速幂法求特征值应用实例。采用瑞利商加速幂法求矩阵 A 的主特征值和主特征向量。

$$A=\begin{bmatrix}1 & 5 & 2\\ 6 & -1 & 7\\ 1 & 3 & 1\end{bmatrix}$$

解： 在 MATLAB 命令窗口中输入：

```
>> A  = [1 5 2; 6 -1 7; 1 3 1];
>> x0 = [ 1 1 1]';
>> [l,v,s]= rpmethod(A,x0)
```

输出计算结果为:

```
l =8.0407
v =1.2239
   1.4131
   0.7759
s = 74
```

由计算结果可知，经过 74 步迭代，求出了矩阵 A 的主特征值为 8.0407，对应的特征向量为$[1.2239，1.4131，0.7759]^T$。

和幂法相比，同一个矩阵，瑞利商加速幂法少用 10 步得到了主特征值。

6.4.4 收缩法

收缩法可用来求矩阵所有的特征值。它是基于幂法的求解特征值的方法。

设矩阵 A 的 n 个特征值按模从大到小排序为:

$$|\lambda_1|>|\lambda_2|>|\lambda_3|\geqslant\cdots\geqslant|\lambda_n|$$

其相应的 n 个线性无关特征向量为α_1，α_2，…，α_n。在计算 A 的最大特征值 λ_1 及相应特征向量α_1后，可以通过收缩方法，继续用乘幂法计算 λ_2 及其相应的特征向量α_2。具体操作过程介绍如下:

❶ 用幂法求出 A 的主特征值 λ_1 和主特征向量α_1;

❷ 令

$$B=A-\alpha_1A_1^T=\begin{pmatrix} 0 & 0 & \cdots & 0 \\ a_{21}-\alpha_{21}a_{11} & a_{22}-\alpha_{21}a_{12} & \cdots & a_{2n}-\alpha_{21}a_{1n} \\ \vdots & \vdots & & \vdots \\ a_{n1}-\alpha_{n1}a_{11} & a_{n2}-\alpha_{n1}a_{12} & \cdots & a_{nn}-\alpha_{n1}a_{1n} \end{pmatrix}$$

其中 A_1 代表矩阵 A 的第一行组成的列向量，a_{n1} 代表主特征向量α_1的第 n 个分量;

❸ 去掉 A_1 的第 1 行和第 1 列:

$$B_1=\begin{pmatrix} a_{22}-\alpha_{21}a_{12} & a_{23}-\alpha_{21}a_{13} & \cdots & a_{2n}-\alpha_{21}a_{1n} \\ a_{32}-\alpha_{31}a_{12} & a_{33}-\alpha_{31}a_{13} & \cdots & a_{3n}-\alpha_{31}a_{1n} \\ \vdots & \vdots & & \vdots \\ a_{n2}-\alpha_{n1}a_{12} & a_{n3}-\alpha_{n1}a_{13} & \cdots & a_{nn}-\alpha_{n1}a_{1n} \end{pmatrix}$$

则 B_1 与有与 A 除 λ_1 外的相同的 $n-1$ 个特征值$|\lambda_2|>|\lambda_3|\geqslant\cdots\geqslant|\lambda_n|$，可以用幂法计算 λ_2 及其对应的特征向量α_2，如此经过 n 次收缩，就可把 A 的所有特征值求出来。

在 MATLAB 中编程实现的收缩法的函数为: spmethod。

功能: 收缩法求矩阵全部特征值。

调用格式: T=spmethod (A, eps)。

其中，A 是已知矩阵;

eps 是精度;

T 是对角阵，对角元素为矩阵 A 的特征值。

收缩法的 MATLAB 程序代码如下所示：

```
function T=spmethod(A,eps)
%收缩法求矩阵全部特征值
%已知矩阵：A
%精度：eps
%求得的矩阵特征值：T
if nargin==1
   eps = 1.0e-6;
end
N = size(A);
n = N(1,1);
T = zeros(n,n);
B = A;
B1= B;
for(i=1:n)
   s  = size(B);
   r  = s(1,1);                                  %r 为 B 的阶
   [T(i,i),u] = pmethod(B,ones(r,1),eps);   %用幂法求解主特征值和主特征向量
   u = u/u(1,1);                                 %将主特征向量规一化
   B1 = B - u*B(1,:);
   s1  = size(B1);
   r1  = s(1,1);
   B  = B1(2:r1,2:r1);                          %B 为收缩后的矩阵
end
```

例 6-7 收缩法求特征值应用实例。采用收缩法求矩阵 A 的所有特征值。

$$A=\begin{bmatrix}3 & 2 & 1\\ 4 & 3 & 7\\ 5 & 8 & 6\end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A = [3 2 1; 4 3 7; 5 8 6];
>> T = spmethod(A)
```

输出计算结果为：

```
T =13.3751        0         0
        0   -3.1534         0
        0         0    1.7782
```

所以用收缩法得到矩阵 A 的三个特征值为 13.3751，–3.1534，1.7782。



在用以上几种方法求矩阵的特征值时，如果矩阵为病态矩阵或者矩阵有复特征值，就会出现错误的结果。

6.4.5 逆幂法

逆幂法是用来求矩阵按模的最小特征值及其对应的特征向量的迭代法。逆幂法的迭代公式如下所示。

给定迭代初始值 $x^0 \neq 0$和误差限$\varepsilon > 0$，

$$Ay^{k+1} = x^k$$

$$m^{k+1} = \max(y^{k+1})$$

$$x^{k+1} = \frac{y^{k+1}}{m^{k+1}}$$

如果$\left|(m^{k+1} - m^k)\right| < \varepsilon$，则结束。

其中，$\max(y^{k+1})$ 表示向量 y^{k+1} 按模最大的分量。

逆幂法有如下的收敛性质：

$$\lim_{k\to\infty} x^k = a_n \big/ \max(a_n), \quad \lim_{k\to\infty} m^k = \frac{1}{\lambda_n}$$

其中 λ_n 为矩阵按模的最小特征值，a_n 为 λ_n 对应的特征向量。

> 在 MATLAB 中编程实现的逆幂法的函数为：ipmethod。
> 功能：收缩法求矩阵全部特征值。
> 调用格式：[l, v]=ipmethod (A, x0, eps)。
> 其中，A 为已知矩阵；
> x0 为迭代初始向量；
> eps 为迭代精度；
> l 为求得的矩阵按模的最小特征值；
> v 为矩阵特征值 l 对应的特征向量。

逆幂法的 MATLAB 程序代码如下所示：

```
function [l,v]=ipmethod(A,x0,eps)
%逆幂法求矩阵按模的最小特征值及其特征向量
%已知矩阵：A
%迭代初始向量：x0
%迭代的精度：eps
%求得的矩阵特征值：l
%求得的矩阵特征值 l 对应的特征向量：v
if nargin==2
    eps = 1.0e-6;
end
v = x0;                    %v 按模的最小特征值对应的特征向量
M = 500;                   %迭代步数限制
m = 0;
l = 0;
for(k=1:M)
```

```
    y = A\v;
    m = max(y);            %m 为按模最大的分量
    v = y/m;
    if(abs(m - 1)<eps)
        l = 1/m;           %达到所需精度，则退出，l 为按模的最小特征值
        return;
    else
        if(k==M)
            disp('迭代步数太多，收敛速度太慢！');
            l = 1/m;
        else
            l = m;
        end
    end
end
```

例 6-8 逆幂法求特征值应用实例。采用逆幂法求矩阵 A 的特征值。

$$A=\begin{bmatrix} 1 & 2 & -2 \\ -4 & 3 & 0 \\ 5 & 1 & 4 \end{bmatrix}$$

解： 在 MATLAB 命令窗口中输入：

```
>> A = [1 2 -2; -4 3 0; 5 1 4];
>> x0 = [1 1 1]';
>> [l,v] = ipmethod(A,x0)
```

输出计算结果为：

```
l = 4.1129
v =-0.1946
    0.6967
    1.0000
```

所以得到 A 的按模最小的特征值为 4.1129，其对应的特征向量为[–0.1946，0.6967，1.0000]^T。

6.4.6 位移逆幂法

位移逆幂法是用来求矩阵离某个特定的常数最近的特征值及其对应的特征向量的迭代法。位移幂法的迭代公式如下所示。

给定迭代初始值 $x^0\neq 0$和误差$\varepsilon>0$，常数μ，

$$(A-\mu I)y^{k+1}=x^k$$

$$m^{k+1}=\max(y^{k+1})$$

$$x^{k+1}=\frac{y^{k+1}}{m^{k+1}}$$

如果$|(m^{k+1}-m^k)|<\varepsilon$，结束。

其中，$\max(y^{k+1})$ 表示向量 y^{k+1} 按模最大的分量。

位移逆幂法有如下的收敛性质：

$$\lim_{k\to\infty} x^k = \frac{a_j}{\max(a_j)}, \quad \lim_{k\to\infty} m^k = \frac{1}{\lambda_j - \mu}$$

其中 λ_j 为离 μ 最近的特征值，a_j 为 λ_j 对应的特征向量。

> 在 MATLAB 中编程实现的位移逆幂法的函数为：dimethod。
>
> 功能：位移逆幂法求矩阵离某个常数最近的特征值及其对应的特征向量。
>
> 调用格式：[l, v]=dimethod (A, x0, u, eps)。
>
> 其中，A 为已知矩阵；
>
> x0 为迭代初始向量；
>
> u 为常数；
>
> eps 为迭代精度；
>
> l 为求得的矩阵离 u 最近的特征值；
>
> v 为矩阵特征值 l 对应的特征向量。

位移逆幂法的 MATLAB 程序代码如下所示：

```
function [l,v]=dimethod(A,x0,u,eps)
%位移逆幂法求矩阵离某个常数最近的特征值及其对应的特征向量
%已知矩阵：A
%迭代初始向量：x0
%常数：u
%迭代的精度：eps
%求得的矩阵离 u 最近的特征值：l
%求得的矩阵的特征值 l 对应的特征向量：v
if nargin==3
    eps = 1.0e-6;
end
N = size(A);
n = N(1,1);                %n 为 A 的阶
v = x0;
M = 5000;                  %迭代步数限制
m = 0;
l = 0;
for(k=1:M)
    y = (A-u*eye(n,n))\v;
    m = max(y);            %m 为按模最大的分量
    v = y/m;
    if(abs(m - l)<eps)
        l = 1/m + u;       %到所需精度，退出
        return;
    else
        if(k==M)
            disp('迭代步数太多，收敛速度太慢！');
```

```
            l = 1/m + u;
        else
            l = m;
        end
    end
end
```

例 6-9 位移逆幂法求特征值应用实例。采用位移逆幂法求矩阵 A 的离 4 最近的特征值。

$$A=\begin{bmatrix} 1 & 2 & -2 \\ -4 & 3 & 0 \\ 5 & 1 & 4 \end{bmatrix}$$

解： 在 MATLAB 命令窗口中输入：

```
>> A = [1 2 -2; -4 3 0; 5 1 4];
>> x0 = [1 1 1]';
>> [l,v] = dimethod(A,x0,4)
```

输出计算结果为：

```
l =4.1145
v =-0.1943
    0.6974
    1.0000
```

所以矩阵 A 离 4 最近的特征值为 4.1145。



如果所有特征值离给定的数太远，结果将不会收敛。

6.4.7 QR 算法

QR 算法是求矩阵特征值的最有效和应用最广泛的一种方法，算法的基本依据以下两个定理：

- 设 A 是 n 阶矩阵，其 n 个特征值为 $\lambda_1,\lambda_2,\cdots,\lambda_n$，那么存在一个酉矩阵 U，使得 $U^H AU$ 是以 $\lambda_1,\lambda_2,\cdots,\lambda_n$ 为对角元的上三角矩阵；
- 设 A 是 n 阶实矩阵，那么，存在一个正交矩阵 Q，使得 $Q^H AQ$ 为一个准上三角矩阵，它的每一个对角元是 A 的一个特征值，对角元上的二阶块矩阵的两个特征值是 A 的一对共轭复特征值。

QR 算法的具体形式有很多种，下面讲述常用的 QR 基本算法、海森伯格 QR 算法和位移 QR 算法。

1. QR 基本算法

QR 基本算法的过程介绍如下：

给定循环步数 M， $A^1=A$ ， $k=1,2,\cdots,M$ ，计算：

$$A^k = Q^k R^k, \quad A^{k+1} = R^k Q^k$$

QR 基本算法有如下的收敛性质：

如果A的特征值满足$|\lambda_1|>|\lambda_2|>|\lambda_3|\geqslant\cdots\geqslant|\lambda_n|$，则QR基本算法产生的矩阵序列$\{A^k\}$基本收敛到上三角矩阵（特别地，当 A 为对称阵时，收敛到对角阵），对角元素收敛到 A 的特征值。

> 在 MATLAB 中编程实现的 QR 基本算法的函数为：qrtz。
> 功能：QR 基本算法求矩阵全部特征值。
> 调用格式：T = qrtz (A, M)。
> 其中，A 为已知矩阵；
> M 为迭代步数；
> l 为矩阵 A 的全部特征值。

QR 基本算法的 MATLAB 程序代码如下所示：

```
function l = qrtz(A,M)
%QR 基本算法求矩阵全部特征值
%已知矩阵：A
%迭代步数：M
%求得的矩阵特征值：l
for(i=1:M)                          %M 为迭代步数
    [q,r]=qr(A);
    A = r*q;
    l = diag(A);
end
```

例 6-10 QR 基本算法求特征值应用实例 1。采用 QR 基本算法求矩阵 A 的所有特征值。

$$A=\begin{bmatrix}1 & 5 & 6\\ 4 & 7 & 0\\ 8 & 11 & 4\end{bmatrix}$$

解： 在 MATLAB 命令窗口中输入：

```
>> A = [1 5 6; 4 7 0; 8 11 4];
>> l = qrtz(A,20)    %迭代 20 步计算
```

输出计算结果为：

```
l =13.4695
   -3.8565
    2.3869
```

继续在 MATLAB 命令窗口中输入：

```
>> l = qrtz(A,500)   %迭代 500 步计算
```

输出计算结果为:

```
l =13.4695
   -3.8566
    2.3871
```

因此得到矩阵 A 的三个特征值为 13.4695、–3.8565、2.3869。

从上面可以看出，QR 基本算法的收敛速度很快（20 步时便已收敛），而且用其他方法可能不收敛的矩阵却可以用 QR 基本算法求出其特征值。如果矩阵为病态的话，则用 QR 基本算法也可求出其特征值，但需要迭代的步骤较多。

例 6-11 QR 基本算法求特征值应用实例 2。采用 QR 基本算法求病态矩阵 A 的所有特征值。

$$A = \begin{bmatrix} 1 & 2 & -2 \\ -4 & 3 & 0 \\ 3 & 1 & 4 \end{bmatrix}$$

解: 在 MATLAB 命令窗口中输入:

```
>> A = [1 2 -2; -4 3 0; 3 1 4];
>> [l,v] = pmethod(A,x0)  %幂法求取，最大迭代 5000 步
```

输出计算结果为:

```
（迭代步数太多，收敛速度太慢！）
l =3.6464
v =0.0542
   -1.2547
    1.0000
```

继续在 MATLAB 命令窗口中输入:

```
>> l= qrtz(A,3000)  %QR 基本算法求取，迭代 3000 步
```

输出计算结果为:

```
l =1.3669
    2.5185
    4.1146
```

继续在 MATLAB 命令窗口中输入:

```
>> l = qrtz(A,5000)     %QR 基本算法求取，迭代 5000 步
```

输出计算结果为:

```
l =2.7055
    1.1801
    4.1145
```

用特征多项式法计算可得出 A 的特征值为$[4.1145, 1.9428+3.6386*i, 1.9428-3.6386*i]^T$。上面的计算表明用幂法不收敛，QR 基本算法是不能求复特征值的，如果矩阵为病态矩阵

的话，QR 基本算法需要较多的迭代才能得出实特征根。


如果不满足$|\lambda_1|>|\lambda_2|>|\lambda_3|\geqslant\cdots\geqslant|\lambda_n|$的条件，QR 基本算法可能不收敛。

2. 海森伯格 QR 算法

当 A 为一般矩阵时，QR 基本算法的计算量很大，在实际使用时，通常采用的做法是先将矩阵 A 正交相似变换为上海森伯格矩阵，然后再实施 QR 基本算法。

给定循环步数 M，$A^1=\text{Hessenberg}(A)$， $k=1,2,\cdots,M$ ，计算：

$$A^k=Q^kR^k$$

$$A^{k+1}=R^kQ^k$$

这样每一步迭代过程中的Q^k和A^k都是上海森伯格矩阵。所谓上海森伯格矩阵是指一个 n 阶矩阵 A，如果当 $i>j+1$ 时，$a_{ij}=0$，称 A 为上海森伯格矩阵。

在 MATLAB 中编程实现的海森伯格 QR 算法的函数为：hessqrtz。
功能：海森伯格 QR 算法求矩阵全部特征值。
调用格式：T = hessqrtz (A, M)。
其中，A 为已知矩阵；
M 为迭代步数；
l 为矩阵 A 的全部特征值。

海森伯格 QR 算法的 MATLAB 程序代码如下所示：

```
function l = hessqrtz(A,M)
%海森伯格 QR 算法求矩阵全部特征值
%已知矩阵：A
%迭代步数：M
%求得的矩阵特征值：l
A = hess(A);            %将 A 化为上海森伯格矩阵
for(i=1:M)              %M 为迭代步数
    [q,r]=qr(A);
    A = r*q;
    l = diag(A);
end
```

例 6-12 海森伯格 QR 算法求特征值应用实例。采用海森伯格 QR 算法求矩阵 A 的所有特征值。

$$A=\begin{bmatrix}6&3&2\\4&3&8\\7&9&5\end{bmatrix}$$

解： 在 MATLAB 命令窗口中输入：

```
>> A = [6 3 2; 4 3 8; 7 9 5];
>> l = hessqrtz(A,20)   %迭代 20 步来求取
```

输出计算结果为：

```
l =15.4121
   -4.4150
    3.0029
```

所以得到矩阵 A 的三个特征值为 15.4121、−4.4150、3.0029。而正确结果计算如下。

继续在 MATLAB 命令窗口中输入：

```
>> l = Chapoly(A)
```

输出计算结果为：

```
l = 3.0
    15.412
   -4.4120
```

由此可见海森伯格 QR 算法收敛速度也是很快的。

3. 位移 QR 算法

为了加快 QR 算法的收敛速度，产生了所谓的位移 QR 算法，它类似于位移逆幂法。其算法的迭代过程介绍如下。

给定循环步数 M，$A^1 = \text{Hessenberg}(A)$， $k = 1,2,\cdots,M$ ，选择 μ^k ，然后计算：

$$A^k - \mu^k I = Q^k R^k, \quad A^{k+1} = R^k Q^k + \mu^k I$$

一般 μ^k 的选择有以下两种考虑方法：

- 选 $\mu^k = a_{nn}^k$ ，即瑞利商位移；
- 迭代过程中，当子矩阵 $\begin{bmatrix} a_{n-1,n-1}^k & a_{n-1,n}^k \\ a_{n,n-1}^k & a_{nn}^k \end{bmatrix}$ 的两个特征值为实数时，选最接近 a_{nn}^k 的那个特征值作为 μ^k ，即威尔金森位移。

在 MATLAB 中编程实现的瑞利商位移 QR 算法的函数为：rqrtz。

功能：瑞利商位移 QR 算法求矩阵全部特征值。

调用格式：T = rqrtz (A, M)。

其中，A 为已知矩阵；

　　M 为迭代步数；

　　l 为矩阵 A 的全部特征值。

瑞利商位移 QR 算法的 MATLAB 程序如下所示：

```
function l = rqrtz(A,M)
%瑞利商位移 QR 算法求矩阵全部特征值
%已知矩阵：A
%迭代步数：M
%求得的矩阵特征值：l
A = hess(A);
for(i=1:M)
```

```
    N = size(A);
    n = N(1,1);
    u = A(n,n);
    [q,r]=qr(A-u*eye(n,n));
    A = r*q+u*eye(n,n);
    l = diag(A);
end
```

在 MATLAB 中编程实现威尔金森位移 QR 算法的函数为：wilkqrtz。

功能：威尔金森位移 QR 算法求矩阵全部特征值。

调用格式：T = wilkqrtz (A, M)。

其中，A 为已知矩阵；

M 为迭代步数；

l 为矩阵 A 的全部特征值。

威尔金森位移 QR 算法的 MATLAB 程序如下所示：

```
function l = wilkqrtz(A,M)
%威尔金森位移 QR 算法求矩阵全部特征值
%已知矩阵：A
%迭代步数：M
%求得的矩阵特征值：l
A = hess(A);
for(i=1:M)
    N = size(A);
    n = N(1,1);
    A1 = A((n-1):n,(n-1):n);
    t = Chapoly(A1);
    if(imag(t(1,1)) == 0 && imag(t(2,1)) == 0)  %两个特征值是否
                                                %为实数
        if(abs(t(1,1)-A(n,n))<abs(t(1,1)-A(n,n)))
            u = t(1,1);
        else
            u = t(2,1);        %选最接近 A(n,n)的那个作为 u
        end
    else
        u = A(n,n);            %如果两个特征值有一个为复数,取 A(n,n)作为 u
    end
    [q,r]=qr(A-u*eye(n,n));
    A = r*q+u*eye(n,n);
    l = diag(A);
end
```

例 6-13 位移 QR 算法求特征值应用实例。采用位移 QR 算法求矩阵 A 的所有特征值。

$$\boldsymbol{A}=\begin{bmatrix}6 & 3 & 2\\ 4 & 3 & 8\\ 7 & 9 & 5\end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A = [6 3 2; 4 3 8; 7 9 5];
>> l = rqrtz(A,13)      %威尔金森位移 QR 算法，迭代 13 步
```

输出计算结果为：

```
l =15.4121
    3.0000
   -4.4121
>> l = wilkqrtz(A,13)  %瑞利商位移 QR 算法，迭代 13 步
```

输出计算结果为：

```
l =15.4121
    3.0000
   -4.4121
```

所以得到矩阵 A 的三个特征值为 15.4121、3.0000、–4.4121。通常来说，位移 QR 算法比 QR 基本算法的收敛速度更快。

6.5 舒尔分解和奇异值分解

舒尔分解可以直接给出矩阵的特征值。而对于对称矩阵，奇异值分解能给出特征值的绝对值。这两种分解的函数的用法在第 2 章的第 2.3.2 节中已介绍过，此处不再赘述。

例 6-14 舒尔分解法求特征值应用实例。采用舒尔分解法求矩阵 A 的所有特征值。

$$A=\begin{bmatrix}-4 & 3 & 10\\ 9 & 1 & 8\\ 7 & 6 & 2\end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A = [-4 3 10; 9 1 8; 7 6 2];
>> [S,U] = schur(A)
```

输出计算结果为：

```
S =-0.4422   -0.8713   -0.2130
   -0.6719    0.4791   -0.5647
   -0.5941    0.1066    0.7973
U =13.9957    3.5156    -3.8192
         0    -8.6568   -4.6962
         0         0    -6.3388
```

则 A 的三个特征值为 13.9957、–8.6568、–6.3388。

例 6-15 奇异分解法求特征值应用实例。采用奇异分解法求矩阵 A 的所有特征值。

$$A=\begin{bmatrix}-4 & 9 & 10\\ 9 & 1 & 8\\ 10 & 8 & 2\end{bmatrix}$$

解： 在 MATLAB 命令窗口中输入：

```
>> A = [-4 9 10; 9 1 8; 10 8 2];
>> [S,U] = svd(A)
```

输出计算结果为：

```
S = 0.5249   -0.8505   0.0327
    0.5782    0.3282  -0.7470
    0.6246    0.4110   0.6641
U =  17.8115       0          0
          0   12.3053          0
          0         0     6.5062
```

则 A 的三个特征值的绝对值为 17.8115、12.3053、6.5062，事实上 A 的特征值为 17.8115、−12.3053、−6.5062。

6.6 采用 eig 函数计算

MATLAB 提供了一个求矩阵特征值的函数 eig，eig 函数功能强大，它综合了以上几种求解特征值的方法，而且能求出复特征值。它的用法如表 6-2 所示。

表 6-2　eig 函数的用法

格　式	说　明
L = eig(A)	求 A 的特征值
[V, L] = eig(A)	求 A 的特征值和特征向量
[V, L] = eig(A,'nobalance')	如果 A 中某项非常小，采用'nobalance'选项求 A 的特征值和特征向量要精确一些
[V, L] = eig(A,B)	计算广义特征向量阵和广义特征值阵，使 AV=BVD
[V, L] = eigs(A,k,s)	计算 k 个由 s 指定的 A 的特征向量和特征值

表 6-2 中第 5 式中 s 可取的值如下：

- 'lm'：求具有最大幅值的 k 个特征向量和特征值；
- 'sm'：求具有最小幅值的 k 个特征向量和特征值；
- 'lr'：求具有最大实部的 k 个特征向量和特征值；
- 'sr'：求具有最小实部的 k 个特征向量和特征值；
- 'be'：将特征值的实部按大小顺序排列，取两端共 k 个特征向量和特征值，如果 k 为奇数，实部大的那端比实部小的那端多一个。

例 6-16　eig 函数求特征值应用实例 1。采用 eig 函数求一般矩阵 A 的所有特征值。

$$A=\begin{bmatrix}-1 & 7 & 6\\ 0 & 1 & 5\\ 3 & -8 & 2\end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A = [-1 7 6; 0 1 5; 3 -8 2];
>> [V, L] = eig(A)
```

输出计算结果为：

```
V = -0.9339          -0.7759                  -0.7759
    -0.3490         -0.3779 - 0.2594i    -0.3779 + 0.2594i
    -0.0780          0.3191 - 0.2934i     0.3191 + 0.2934i
L = 2.1177               0                     0
          0        -0.0589 + 4.6093i          0
          0               0                 -0.0589 - 4.6093i
```

由计算结果可知，矩阵 A 的三个特征值为 2.1177、$-0.0589+4.6093i$、$-0.0589-4.6093i$。

例 6-17 eig 函数求特征值应用实例 2。采用 eig 函数求病态矩阵 A 的所有特征值。

$$A=\begin{bmatrix}1 & 10^{-18} & 2\\ 2.0\times10^{-18} & 6 & 3\\ 5 & 2.0\times10^{-19} & 4\end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A=[1 1.0e-18  2 ; 2.0e-18 6 3;5  2.0e-19  4];
>> [V, L] = eig(A)
```

输出计算结果为：

```
V = -0.6767    0.0000    0.0000
    -0.2900    1.0000   -1.0000
     0.6767    0.0000    0.0000
L = -1.0000         0         0
          0    6.0000         0
          0         0    6.0000
```

继续在 MATLAB 命令窗口中输入：

```
>> [V,L]=eig(A,'nobalance')
```

输出计算结果为：

```
V = -1.0000             0.0000 + 0.0000i   0.0000 - 0.0000i
    -0.4286            -0.0000 - 1.0000i  -0.0000 + 1.0000i
     1.0000             0.0000 + 0.0000i   0.0000 - 0.0000i
L = -1.0000                  0                 0
          0             6.0000 + 0.0000i       0
          0                  0                 6.0000 - 0.0000i
```

所以矩阵 A 的三个特征值为 -1.0000、$6.0000+0.0000i$、$6.0000-0.0000i$。

从上面的例子可以看出，'nobalance'选项可提高结果的精度。

例 6-18 eig 函数求特征值应用实例 3。采用 eigs 函数求矩阵 A 的任意个数特征值。

$$A=\begin{bmatrix} 1 & 5 & 2 \\ -7 & 6 & 3 \\ 5 & 9 & 4 \end{bmatrix}$$

解： 在 MATLAB 命令窗口中输入：

```
>> A=[1 5  2 ; -7 6 3; 5  9  4];
>> [V,L]=eigs(A,2,'lm')  %求矩阵 A 的具有最大幅值的 2 个特征值及对应的特征向量
```

输出计算结果为：

```
V =0.3830    0.2623
   0.0854   -0.2811
   0.9198    0.9231
L =    6.9173         0
         0          2.6804
```

继续在 MATLAB 命令窗口中输入：

```
>> [V,L]=eigs(A,2,'sr')  %求矩阵 A 的具有最小实部的 2 个特征值及对应的特征向量
```

输出计算结果为：

```
V =-0.1612    0.2623
   0.3553   -0.2811
  -0.9207    0.9231
L =    1.4023         0
         0          2.6804
```

从计算结果可以看出，矩阵 A 的三个实特征值从小到大排列为 1.4023、2.6804 与 6.9173。

6.7 矩阵指数计算

矩阵级数 $e^{At}=\sum_{n=0}^{\infty}\frac{(At)^n}{n!}$ 称为矩阵指数。

如果按定义来求矩阵指数，显然运算量非常大。利用下面的两条性质可以很快算出一些矩阵的矩阵指数：

- 如果矩阵 A 为对角阵 $\mathrm{diag}(\lambda_1,\lambda_2,\cdots,\lambda_n)$，则它的矩阵指数为 $\mathrm{diag}(e^{\lambda_1},e^{\lambda_2},\cdots,e^{\lambda_n})$；
- $\exp(P^{-1}AP)=P^{-1}\exp(A)P$。

因此要求一个矩阵的矩阵指数，可以先求其特征值，再用第二条性质来求其矩阵指数。

例 6-19 矩阵指数求取实例。求矩阵 $A=\begin{bmatrix} 3 & 2 & 6 \\ 1 & 0 & 5 \\ -4 & 8 & 9 \end{bmatrix}$ 的矩阵指数。

解： 在 MATLAB 命令窗口中输入以下命令：

```
>> A=[3 2 6;1 0 5; -4 8 9];
>> [V,L]=eig(A)
V = 0.6767   -0.9129    0.2283
    0.3917   -0.3651    0.8380
    0.6234   -0.1826   -0.4956
L =9.6847         0          0
   0    5.0000         0
   0         0   -2.6847
>> syms t;
>> B=V*exp(L*t)*inv(V);
>> B=vpa(B,6)   %结果保留 6 位有效数字
```

由于 MATLAB 输出结果很长，将其整理为如下形式：

$$e^{At}=\begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix}$$

其中：

M_{11} =−0.575869*exp(9.68466*t)−0.975788+1.66667*exp(5*t)−0.0907977*exp(−2.68466*t);

M_{12} =0.851988*exp(9.68466*t)−0.02844−1.11111*exp(5*t)+0.259123*exp(−2.68466*t);

M_{13} =1.17537*exp(9.68466*t)−0.02104−1.11111*exp(5*t)−0.0642577*exp(−2.68466*t);

M_{21} =−0.333333*exp(9.68466*t)−2.65813+0.666669*exp(5*t)−0.333333*exp(−2.68466*t);

M_{22} =0.493161*exp(9.68466*t)+2.12226−0.444445*exp(5*t)+0.951284*exp(−2.68466*t);

M_{23} =0.680345*exp(9.68466*t)+2.31059−0.444446*exp(5*t)−0.235901*exp(−2.68466*t);

M_{31} =−0.530470*exp(9.68466*t)+0.168500+0.333333*exp(5*t)+0.197137*exp(−2.68466*t);

M_{32} =0.784821*exp(9.68466*t)−0.197920−0.222222*exp(5*t)−0.562599*exp(−2.68466*t);

M_{33} =1.08271*exp(9.68466*t)−1.14647−0.222222*exp(5*t)+0.139514*exp(−2.68466*t);

这样就轻松计算出了矩阵 A 的矩阵指数。

6.8 小结

本章详细讲述了 MATLAB 中进行矩阵特征值运算的常用解法，包括特征多项式法、幂法、收缩法、逆幂法和 QR 方法等。

进行矩阵特征值计算，用幂法求最大模特征值特别适用于稀疏矩阵，它计算简单，如果用逆幂法结合位移技巧则可以加快收敛速度；QR 方法可以求全部的特征值，它对于中小型稠密矩阵的特征值计算非常有效，特别对于对称矩阵，QR 算法简直可以说是完美的算法；此外，一些古典的方法如雅克比方法，不但过程十分复杂，而且与现代的一些方法相比无明显的优越性，因此本章没有介绍。

第 7 章　数值微分

微分和积分是非常重要的数学运算，本章讨论微分的数值算法。对于常见的初等函数，如幂函数、指数函数和三角函数等，可以通过解析的方法来求出其导数，但是对于一些超越函数，如级数形式的函数等，就很难直接微分了，甚至有时候函数的表达式都不知道，只获得函数的一系列离散点，如实验数据，在这样的情况下求函数的导数就不能依靠函数的表达式了，而是要依靠本章所介绍的几种数值方法。

通过本章，读者不仅能掌握常见的数值微分算法，而且还能熟练使用 MATLAB 编程来实现这些算法。

7.1 中点公式法

此方法的思想十分简单，由导数的定义知：

$$f'(x)=\lim_{h\to 0}\frac{f(x+h)-f(x-h)}{2h}$$

在实际应用中，可以用有限的很小的h值代入右边的式子求得导数的近似值。表面上看，似乎步长越小，结果越精确，其实不然，当h值十分小时，$f(x+h)$的值与$f(x-h)$的值很接近，在计算机上直接相减会造成有效数字的损失，因此h值也不能太小。

一般情况下，如果$f(x)$足够光滑，h取 0.1 就够了。

在 MATLAB 中编程实现的中点公式法求一阶导数的函数为：MidPoint。
功能：中点公式法求取一阶导数。
调用格式：df=MidPoint(func,x0,h)。
其中，func：函数名；
　　x0：求导点；
　　h：迭代步长；
　　df：导数值。

中点公式法的 MATLAB 程序代码如下所示：

```
function df=MidPoint(func,x0,h)
%采用中点公式法，求取函数 func 在 x0 处的导数
%函数名：func
% 求导点：x0
% 迭代步长 ：h
```

```
% 导数值: df
if nargin == 2
   h = 0.1;                                        %h 默认为 0.1
else if (nargin == 3 && h == 0.0)
       disp('h 不能为 0! ');
       return;
   end
end
y1 = subs(sym(func), findsym(sym(func)),x0+h);
y2 = subs(sym(func), findsym(sym(func)),x0-h);
df = (y1-y2)/(2*h);                               %中点公式
```

例 7-1 中点公式法求一阶导数应用实例。采用中点公式法求函数 $f=\sqrt{x}$ 在 $x=4$ 处的导数。

解：在 MATLAB 命令窗口中输入：

```
>>df=MidPoint('sqrt(x)',4)
```

输出计算结果为：

```
df =    0.2500
```

而函数 $f=\sqrt{x}$ 在 $x=4$ 处导数值的精确值也是 0.25。

7.2 三点公式法和五点公式法

三点公式法是由等距节点插值公式得来的，其思想是先将函数用等距节点公式进行插值，再对插值多项式求导数，再取有限的项数。

三点公式有以下三种形式，分别由等距牛顿前插、等距牛顿后插和斯特林公式得出以下结论。

- 等距牛顿前插得出的三点公式：

$$f'(x_0)\approx\frac{1}{2h}(-3y_0+4y_1-y_2)$$

- 等距牛顿后插得出的三点公式：

$$f'(x_0)\approx\frac{1}{2h}(3y_0-4y_{-1}+3y_{-2})$$

- 斯特林公式得出的三点公式：

$$f'(x_0)\approx\frac{1}{2h}(y_1-y_{-1})$$

上述式子中

$$y_0=f(x_0),y_1=f(x_0+h),y_2=f(x_0+2h)$$
$$y_{-1}=f(x_0-h),y_{-2}=f(x_0-2h)$$

五点公式法和三点公式法的得出过程是一样的，只不过取的项数比三点公式法多：

$$f'(x_0) \approx \frac{1}{12h}(-25y_0 + 48y_1 - 36y_2 + 16y_3 - 3y_4)$$

$$f'(x_0) \approx \frac{1}{12h}(-3y_{-1} - 10y_0 + 18y_1 - 16y_2 + y_3)$$

$$f'(x_0) \approx \frac{1}{12h}(y_{-2} - 8y_{-1} + 8y_1 - y_2)$$

$$f'(x_0) \approx \frac{1}{12h}(3y_1 + 10y_0 - 18y_{-1} + 16y_{-2} - y_{-3})$$

$$f'(x_0) \approx \frac{1}{12h}(25y_0 - 48y_{-1} + 36y_{-2} - 16y_{-3} + 3y_{-4})$$

在 MATLAB 中编程实现的三点公式法求一阶导数的函数为：ThreePoint。

功能：三点公式法求函数的一阶导数。

调用格式：df= ThreePoint (func,x0,type,h)。

其中，func：函数名；

x0：求导点；

type：采用的三点公式法的形式；

h：步长；

df：导数值。

三点公式法的 MATLAB 程序代码如下所示：

```
function df=ThreePoint(func,x0,type,h)
%采用三点公式法求函数 func 在 x0 处的导数
%函数名： func
%求导点： x0
%三点公式法的形式：  type
%步长 ： h
%导数值： df
if nargin == 3
   h = 0.1;
else if (nargin == 4 && h == 0.0)
        disp('h 不能为 0! ');
        return;
   end
end
for i=1:5
    y(i) = subs(sym(func), findsym(sym(func)), x0-2*h+i*h-h);
end
hd = 1/2/h;
switch type
   case 1,
      df = (-3*y(3)+ 4*y(4) -y(5))*hd;      %用第一个公式求一阶导数
   case 2,
      df = (3*y(3) - 4*y(2) + 3*y(1))*hd;  %用第二个公式求一阶导数
   case 3,
```

```
        df = (y(4) - y(2))*hd;                      %用第三个公式求一阶导数
    end
```

在 MATLAB 中编程实现的五点公式法求一阶导数的函数为：FivePoint。

功能：五点公式法求函数的一阶导数。

调用格式：df= FivePoint (func,x0,type,h)。

其中，func：函数名；

x0：求导点；

type：采用的五点公式法的形式；

h：步长；

df：导数值。

五点公式法的 MATLAB 程序代码如下所示：

```
function df= FivePoint(func,x0,type,h)
%采用五点公式法求函数 func 在 x0 处的导数
%函数名： func
%求导点： x0
%五点公式法的形式： type
%步长 ：h
%导数值： df
if nargin == 3
   h = 0.1;
else if (nargin == 4 && h == 0.0)
       disp('h 不能为 0! ');
       return;
   end
end
for i=1:9
    y(i) = subs(sym(func), findsym(sym(func)), x0-4*h+i*h-h);
end
hd = 1/12/h;
switch type
   case 1,
    df = (-25*y(5)+48* y(6)-36*y(7)+16* y(8)-3*y(9))*hd;%用第一个公式求导数
   case 2,
    df = (-3*y(4)-10* y(5)+18*y(6)-16* y(7)+y(8))*hd; %用第二个公式求导数
   case 3,
    df = (y(3)-8* y(4)+8*y(6)- y(7))*hd;      %用第三个公式求导数
   case 4,
    df = (3*y(2)+10* y(3)-18* y(4)+16* y(5)-y(6))*hd;     %用第四个公式求导数
   case 5,
    df = (25*y(5)-48*y(4)+36* y(3)-16* y(2)+3* y(1))*hd;%用第五个公式求导数
end
```

例 7-2 三点公式法求一阶导数应用实例。三点公式法求函数 $f = \sin(x)$ 在 $x = 2$ 处的导数。

解： 在 MATLAB 命令窗口中输入：

```
>> df1=ThreePoint('sin(x)',2,1);
>> df2=ThreePoint('sin(x)',2,2);
>> df3=ThreePoint('sin(x)',2,3);
```

用三种方法得到的计算结果为：

```
df1 =   -0.4178
df2 =   -0.4173
df3 =   -0.4155
```

而函数 $f=\sin(x)$ 在 $x=2$ 处的导数为 $\cos(2)=-0.4161$，从上面三种方法得出的结果来看，都是十分接近的，其中以第三种方法的近似效果最佳。

例 7-3 五点公式法求一阶导数应用实例。采用五点公式法求函数 $f=\sin(x)$ 在 $x=2$ 处的导数。

解： 在 MATLAB 命令窗口中输入：

```
>> df1=FivePoint('sin(x)',2,1);
>> df2=FivePoint('sin(x)',2,2);
>> df3=FivePoint('sin(x)',2,3);
>> df4=FivePoint('sin(x)',2,4);
>> df5=FivePoint('sin(x)',2,5);
```

用五种方法得到的计算结果为：

```
df1 =   -0.4161
df2 =   -0.4161
df3 =   -0.4161
df4 =   -0.4161
df5 =   -0.4161
```

而函数 $f=\sin(x)$ 在 $x=2$ 处的导数为 $\cos(2)=-0.4161$，从上面的结果来看，五点公式法的精度是很高的。

7.3 三次样条函数法

用三次样条求已知函数 $f(x)$ 在点 x_0 处的导数的算法介绍如下。

❶ 以 x_0 为中心向两边等分出 n（n 为正整数）份，等分间距为 h，即形成下面的分割：

$$x_0-nh<\cdots<x_0-h<x_0<x_0+h<\cdots<x_0+nh$$

令 $x_{-n}=x_0-nh$；

❷ 以这些点形成三次样条函数 S，求出 S 的系数：

$$S(x)=\sum_{j=-1}^{n+1}c_j\Omega_3(\frac{x-x_{-n}}{h}-j)$$

其中 $\Omega_3(\frac{x-x_0}{h}-j)$ 为 B 样条函数；

❸ 按下面的公式求出点 x_0 处的导数：

$$f'(x_0)\approx\frac{1}{2h}(c_{n+1}-c_{n-1})$$

在 MATLAB 中编程实现的三次样条法求导数的函数为：DiffBSample。

功能：三次样条法求函数的导数。

调用格式：df= DiffBSample (func,x0,n,h)。

其中，func：函数名；

x0：求导点；

n：等分份数；

h：步长；

df：导数值。

三次样条法的 MATLAB 程序代码如下所示：

```
function df0 = DiffBSample(func,x0,n,h)
%采用三次样条法求函数 func 在 x0 处的导数
%函数名： func
%求导点： x0
%等分份数：  n
%步长 ： h
%导数值：  df0
format long;
node_num = 2*n + 1;
for i=1:node_num
    y(i) = subs(sym(func), findsym(sym(func)), x0-n*h+i*h-h);
end
y_1 = (-3*y(1)+4*y(2)-y(3))/(2*h);
y_N = (3*y(2*n+1)-4*y(2*n)+3*y(2*n-1))/(2*h);
c = SubBSample(h,2*n,y,y_1,y_N);
df0 = (c(n+3)-c(n+1))/2/h;
format short;
function c = SubBSample(h,n,y,y_1,y_N)
f0 = 0.0;
c = zeros(n+3,1);
b = zeros(n+1,1);
A = diag(4*ones(n+1,1));
I = eye(n+1,n+1);
AL = [I(2:n+1,:);zeros(1,n+1)];
AU = [zeros(1,n+1);I(1:n,:)];
A = A+AL+AU;                    %形成系数矩阵
for i=2:n
   b(i,1) = 6*y(i);
end
b(1) = 6*y(1)+2*h*y_1;
b(n+1) = 6*y(n+1)-2*h*y_N;
d = followup(A,b);          %用追赶法求出系数
c(2:n+2) = d;
```

```
c(1) = c(2) - 2*h*y_1;        %c(-1)
c(n+3) = c(3)+2*h*y_N;        %c(n+1)
```

例 7-4 三次样条法求一阶导数应用实例。三次样条法求函数 $f=\ln x$ 在 $x=4$ 处的一阶导数。

解：在 MATLAB 命令窗口中输入：

```
>> df= DiffBSample ('log(x)',4,10,0.1);       %两边各取 10 个点，步长为 0.1
```

计算结果为：

```
df =    0.2500
```

而 $f=\ln x$ 在 $x=4$ 处的导数的理论值为 0.25。

7.4 自适应数值微分法

用自适应数值微分法求已知函数 $f(x)$ 在点 x_0 处的导数计算方法介绍如下。

首先令 $f_{(1)}'(x_0)=\infty$；从 $k=2,3\cdots,n$ 开始循环迭代，直至收敛，

❶ 以 x_0 为中心向两边等分出 n（n 为正整数）份，等分间距为 h，即形成下面的分割：

$$x_0-nh<\cdots<x_0-h<x_0<x_0+h<\cdots<x_0+nh$$

❷ 以这些点形成三次样条函数 S，求出 S 的系数：

$$S(x)=\sum_{j=-1}^{n+1}c_j\Omega_3(\frac{x-x_{-n}}{h}-j)$$

其中 $x_{-n}=x_0-nh$。

❸ 按下面的公式求出点 x_0 处的导数：

$$f_{(k)}'(x_0)\approx\frac{1}{2h}(c_{n+1}-c_{n-1})$$

❹ 如果 $\left|f_{(k)}'(x_0)-f_{(k-1)}'(x_0)\right|<\varepsilon$（给定精度），则退出；否则令 $h=h*0.75,n=n+4$，转❶。

在 MATLAB 中编程实现的自适应法求导数的函数为：SmartDF。

功能：自适应法求函数的导数。

调用格式：df= SmartDF(func,x0,eps)。

其中，func：函数名；

x0：求导点；

eps：导数精度。

自适应法的 MATLAB 程序代码如下所示：

```
function df0 = SmartDF(func,x0,eps)
%采用自适应法求函数 func 在 x0 处的导数
```

```
%函数名: func
%求导点: x0
%等分份数: n
%步长 : h
%导数值: df0
format long;
if nargin == 2
   eps = 1.0e-4;
end
df0_tmp = Inf;
n = 2;
h = 0.1;
con = 1;
while con
   node_num = 2*n + 1;
   hd = 1/2/h;
   x = (x0-n*h):h:(x0+n*h);
   for i=1:node_num
      y(i) = subs(sym(func), findsym(sym(func)), x(i));
   end
   y_1 = (-3*y(1)+4*y(2)-y(3))*hd;
   y_N = (3*y(node_num)-4*y(node_num-1)+3*y(node_num-2))*hd;
   c = SubBSample(h,2*n,y,y_1,y_N);
   df0 = (c(n+2)-c(n))*hd;
   if abs(df0 - df0_tmp) <= eps
      con = 0;
   else
      df0_tmp = df0;
      n = n+4;
      h = h*0.75;
   end
end
function c = SubBSample(h,n,y,y_1,y_N)
format long;
f0 = 0.0;
c = zeros(n+3,1);
b = zeros(n+1,1);
A = diag(4*ones(n+1,1));
I = eye(n+1,n+1);
AL = [I(2:n+1,:);zeros(1,n+1)];
AU = [zeros(1,n+1);I(1:n,:)];
A = A+AL+AU;                        %形成系数矩阵
b(2:n,1) = 6*y(2:n);
b(1) = 6*y(1)+2*h*y_1;
b(n+1) = 6*y(n+1)-2*h*y_N;
d = followup(A,b);                  %用追赶法求出系数
c(2:n+2) = d;
c(1) = c(2) - 2*h*y_1;              %c(-1)
c(n+3) = c(3)+2*h*y_N;              %c(n+1)
```

例 7-5 自适应法求一阶导数应用实例。采用自适应法求函数 $f=\ln x$ 在 $x=4$ 处的导数。

解：在 MATLAB 命令窗口中输入：

```
>> df= SmartDF('log(x)',4);
```

计算结果为：

```
df =   0.25026424861455
```

这是在默认精度下得出的导数值，进一步提高精度可得出更好的结果：

```
>> df = SmartDF('log(x)',4,1.0e-6)
df =   0.25000264568820
```

上面的计算结果精度为 1.0e-6。

而下面的计算结果的精度则达到了 1.0e-8。

```
>> df = SmartDF('log(x)',4,1.0e-8)
df =   0.25000002602984
```

7.5 辛普森数值微分法

辛普森数值微分是用来求等距节点在节点处的导数的，辛普森数值微分公式如下所示：

$$\begin{bmatrix} 4 & 1 & & & \\ 1 & 4 & 1 & & \\ & 1 & 4 & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & 4 \end{bmatrix}\begin{bmatrix} f'(x_1) \\ f'(x_2) \\ f'(x_3) \\ \vdots \\ f'(x_{n-1}) \end{bmatrix}=\begin{bmatrix} 3(y_2-y_0)/h-f'(x_0) \\ 3(y_3-y_1)/h \\ 3(y_4-y_2)/h \\ \vdots \\ 3(y_n-y_{n-2})/h-f'(x_n) \end{bmatrix}$$

其中，$y_n=f(x_n), x_n=x_0+nh$。

如果端点导数值 $-f'(x_0)$ 和 $-f'(x_n)$ 未知，则将它们用中点微分公式近似，这时的辛普森数值微分公式为：

$$\begin{bmatrix} 2 & 0 & & & \\ 1 & 4 & 1 & & \\ & 1 & 4 & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 0 & 2 \end{bmatrix}\begin{bmatrix} f'(x_1) \\ f'(x_2) \\ f'(x_3) \\ \vdots \\ f'(x_{n-1}) \end{bmatrix}=\begin{bmatrix} (y_2-y_0)/h \\ 3(y_3-y_1)/h \\ 3(y_4-y_2)/h \\ \vdots \\ (y_n-y_{n-2})/h \end{bmatrix}$$

在 MATLAB 中编程实现的辛普森数值微分法法求一阶导数的函数为：CISimpson。

功能：辛普森数值微分法求函数的一阶导数。

调用格式 1：df= CISimpson (func,x0,n,h)。

其中，func：函数名；

x0：求导点；

n：等分份数；
h：步长；
df：导数值。
调用格式 2：df= DISimpson(X,Y,n,p)。
其中，X：横坐标点；
Y：纵坐标点；
n：点的个数；
p：求导点的点编号；
df：导数值。

辛普森数值微分法（适用于已知函数表达式）的 MATLAB 程序代码如下所示：

```
function df=CISimpson(func,x0,n,h)
%采用辛普森数值微分法求函数 func 在 x0 处的导数
%函数名：func
%求导点：x0
%等分份数： n
%步长 ：h
%导数值： df
if nargin == 2                          %以下是参数的判断过程
    h = 0.1;
    n = 5;
else
    if (nargin == 3 )
        if (n<5)
            disp('n 不能小于 5!');
            return;
        else
            h = 0.1;
        end
    else (nargin == 4 && h == 0.0)
        disp('h 不能为 0! ');
        return;
    end
end
for(i=1:n)                              %这个循环计算节点的函数值
     if (mod(n,2) == 0)
        y(i)= subs(sym(func), findsym(sym(func)),x0+(i-n/2)*h);
    else
        y(i)= subs(sym(func), findsym(sym(func)),x0+(i-(n+1)/2)*h);
     end
end
f(1)=(y(3)-y(1))/(2*h);
f(2)=(y(n)-y(n-2))/(2*h);               %这两行用中心微分法给出端点的导数
b(1:n-2,1) = zeros(n-2,1);
b(1,1)=3*(y(3)-y(1))/h-f(1);
b(n-2,1)=3*(y(n)-y(n-2))/h-f(2);
```

```
for(i=2:(n-3))
    b(i,1) = 3*(y(i+2)-y(i))/h;
end                                          %这一块是辛普森公式的右边的列向量
for(i=1:n-2)
    for(j=1:n-2)
        if( (i == j+1) || (j == i+1))
            A(i,j)= 1;
        else if( i == j)
            A(i,j) = 4;
            end
        end
    end
end                                          %这一块是系数矩阵
[Q,R]=qr(A);
DF = R\(Q\b);                                %用 QR 分解法求解
if( mod(n,2) == 0)
    df = DF(n/2);
else
    df = DF((n+1)/2);
end                                          % 求出 x0 处的导数值
```

辛普森数值微分法（适用于数据向量）的另一个 MATLAB 程序代码如下所示：

```
function df = DISimpson(X,Y,n,p)
%横坐标点: X;
%纵坐标点: Y;
%点的个数: n ;
%求导点的点编号: p;
%导数值: df
if n < 5
    disp('n 不能小于 5!');
    return;
end                                          %n 为数据点个数, p 为要求导数值的数据点编号
if p == 0
    disp('p 不能等于 0!');
    return;
end
h = X(2)-X(1);
xx =linspace(X(1),X(n),h);
if(xx ~= X)
    disp('节点之间不是等距的! ');
    return;
end                                          %判断是否是等距节点
f(1)=(Y(3)-Y(1))/(2*h);
f(2)=(Y(n)-Y(n-2))/(2*h);                    %这两行用中心微分法给出端点的导数
b(1,1)=3*(Y(3)-Y(1))/h-f(1);
b(n-2,1)=3*(Y(n)-Y(n-2))/h-f(2);
for(i=2:n-3)
    b(i,1) = 3*(Y(i+2)-Y(i))/h;
end                                          %这一块是辛普森公式的右边的列向量
```

```
for(i=1:n-2)
    for(j=1:n-2)
        if( (i == j+1) || (j == i+1))
             A(i,j) = 1;
        else if( i == j)
             A(i,j) = 4;
            end
        end
    end
end                                                  %这一块是系数矩阵
[Q,R]=qr(A);
DF = R\(Q\b);                                        %用 QR 分解法求解
if( p == 1)
    df = f(1);
else
    df = DF(p-1);                                    % 求出第 p 个节点处的导数值
end
```

例 7-6 辛普森数值微分法应用实例 1。辛普森数值微分法求函数 $f=e^x$ 在 $x=2.5$ 处的导数值。

解： 在 MATLAB 命令窗口中输入：

```
>> df = CISimpson('exp(x)',2.5)          %默认在 x=2.5 左右各取 5 个节点，步长为 0.1
```

输出计算结果为：

```
df =   13.8179
>> df = CISimpson('exp(x)',2.5,10)       %在 x=2.5 左右各取 10 个节点，步长为 0.1
```

输出计算结果为：

```
df =   13.4528
>> df = CISimpson('exp(x)',2.5,100)      %在 x=2.5 左右各取 100 个节点，步长为 0.1
```

输出计算结果为：

```
df =   13.4637
```

而函数 $f=e^x$ 在 $x=2.5$ 处的导数值为 $e^{2.5}=12.1825$。

辛普森数值微分公式的缺点在于：如果端点的导数没有给定，则它的精度会比较低，当然如果端点的导数未知，可以采用其他精度更高的方法来近似端点导数值，上面列出的代码是采用中点微分公式来近似的，精度显然比较低。

例 7-7 辛普森数值微分法应用实例 2。用辛普森数值微分法求下列数据点在 $x=1$ 处的导数值。

x	0	0.5	1	1.5	2	2.5	3
y	0	0.125	1	3.375	8	15.625	27

解： 在 MATLAB 命令窗口中输入：

```
>> x=0:0.5:3;
>> y=[0 0.125 1 3.375 8 15.625 27];
>> df=DISimpson(x,y,7,3)  %求第 3 个节点处的导数（总共 7 个节点）
```

输出计算结果为：

```
df =    3.0308
```

表格中的数据点是根据函数 $f = x^3$ 来离散的，$f = x^3$ 在 $x=1$ 处的导数为 3.00，辛普森数值微分法的结果还是比较好的。

7.6 理查森外推算法

理查森外推算法的迭代公式如下所示：

$$G_1(h) = \frac{f(x+h/2) - f(x-h/2)}{h}$$

$$G_{n+1}(h) = \frac{G_n(\frac{h}{2}) - (\frac{1}{2})^{2n} G_n(h)}{1-(\frac{1}{2})^{2n}}, \quad n = 1,2,\cdots$$

当迭代步数充分大时，$G(h)$ 收敛到 $f'(x)$。此算法是一种金字塔式的算法，底层是 $G_1(h), G_1(\frac{h}{2}), \cdots, G_1(\frac{h}{2^n})$；第二层是 $G_2(h), G_2(\frac{h}{2}), \cdots, G_2(\frac{h}{2^{n-1}})$；顶层是 $G_{n-1}(h)$。如表格 7-1 所示。

表 7-1 理查森外推法计算表

$G_1(h)$	$G_1(\frac{h}{2})$	$\cdots$	$G_1(\frac{h}{2^{n-1}})$	$G_1(\frac{h}{2^n})$
$G_2(h)$	$G_2(\frac{h}{2})$		$G_2(\frac{h}{2^{n-1}})$	
$\vdots$	$\vdots$	$\iddots$		
$G_{n-2}(h)$	$G_{n-2}(\frac{h}{2})$			
$G_{n-1}(h)$				

在 MATLAB 中编程实现的理查森外推算法求导数的函数为：Richason

功能：理查森外推算法求函数的导数。

调用格式：df= Richason (func,x0,n,h)。

其中，func：函数名；

x0：求导点；

n：迭代步数；

h：步长；

df：导数值。

理查森外推算法的 MATLAB 程序代码如下所示：

```
function df = Richason(func, x0, n, h)
%理查森外推算法求函数 func 在 x0 处的导数
%函数名: func
%求导点: x0
%迭代步数: n
%步长: h
%导数值: df
if nargin == 3
   h = 1;
else if (nargin == 3 && h == 0.0)
       disp('h 不能为 0! ');
       return;
   end
end
for(i=1:n)
   y1 = subs(sym(func), findsym(sym(func)),x0+h/(2^i));
   y2 = subs(sym(func), findsym(sym(func)),x0-h/(2^i));
   G(i) = 2^(i-1)*(y1-y2)/h;            %求得金字塔的底层值
end
G1 = G;
for(i=1:n-1)
   for(j=(i+1):n)
       G1(j)=(G(j)-(0.5)^(2*i)*G(j-1))/(1-(0.5)^(2*i)); %求得金字塔的每层值
   end
   G = G1;
end
df = G(n);    %顶层值就是所需得导数值
```

例 7-8 理查森外推算法求导数应用实例。用理查森外推算法求函数 $y=2^x$ 在 $x=1$ 处的导数值。

解： 在 MATLAB 命令窗口中输入：

```
>> df =Richason('2^x',1, 8)                %外推 8 层
```

输出计算结果为：

```
df =    1.3863
```

而函数 $y=2^x$ 在 $x=1$ 的导数的准确值为 2*log(2)= 1.3863，经过试验可以发现，对于各种函数，理查森外推算法都能达到很高的精度。

7.7 二阶导数求取法

前面几节介绍了一阶导数的各种求法，实际工程应用中还可能要用到二阶导数，比如说工程中常常遇到的微分方程，基本上都含有二阶导数项，因此下面介绍求二阶导数的两种方法，对于更高阶导数的求法，可以根据基本公式推导出，但是实际上用到高阶导数的地方比较少。

7.7.1 多点公式法

二阶导数的求法与一阶导数的求法基本上是一样的。基本公式如下所示：

$$f''(x) = \lim_{h\to 0}\frac{f(x+h)+f(x-h)-2f(x)}{h^2}$$

利用插值公式还可得出求二阶导数的多点公式法，常用的有三点公式法、四点公式法和五点公式法。

三点公式法求二阶导数的公式如下所示：

$$f''(x_0)\approx\frac{1}{h^2}(y_0-2y_1+y_2)$$
$$f''(x_0)\approx\frac{1}{h^2}(y_{-1}-2y_0+y_1)$$
$$f''(x_0)\approx\frac{1}{h^2}(y_{-2}-2y_{-1}+y_0)$$

其中

$$y_0=f(x_0), y_1=f(x_0+h), y_2=f(x_0+2h)$$
$$y_{-1}=f(x_0-h), y_{-2}=f(x_0-2h)$$

在 MATLAB 中编程实现的三点公式法求二阶导数的函数为：ThreePoint2。

功能：三点公式法求函数的二阶导数。

调用格式：df= ThreePoint2 (func,x0,type,h)。

其中，func：函数名；

x0：求导点；

type：采用的三点公式法的形式；

h：步长；

df：二阶导数值。

三点公式法的 MATLAB 程序代码如下所示：

```
function df= ThreePoint2(func,x0,type,h)
%采用三点公式法求函数 func 在 x0 处的二阶导数
%函数名：func
%求导点：x0
%三点公式法的形式： type
%步长 ：h
%二阶导数值： df
if nargin == 3
   h = 0.1;
else if (nargin == 4 && h == 0.0)
       disp('h 不能为 0! ');
       return;
   end
end
```

```
for i=1:5
    y(i) = subs(sym(func), findsym(sym(func)), x0-2*h+i*h-h);
end
hd = 1/h/h;
switch type
   case 1,
      df = (y(3) - 2*y(4) + y(5))*hd;        %用第一个公式求二阶导数
   case 2,
      df = (y(2) - 2*y(3) + y(4))*hd;        %用第二个公式求二阶导数
   case 3,
      df = (y(1) - 2*y(2) + y(3))*hd;        %用第三个公式求二阶导数
end
```

例 7-9 三点公式法求二阶导数应用实例。用三点公式法求函数 $y=2^x$ 在 $x=2$ 处的二阶导数值。

解： 在 MATLAB 命令窗口中输入：

```
>> df1 = ThreePoint2 ('2^x',2,1,0.01)
>> df2 = ThreePoint2 ('2^x',2,2,0.01)
>> df3 = ThreePoint2 ('2^x',2,3,0.01)
```

输出的计算结果为：

```
df1 =   1.93518706365303
df2 =   1.92181975020134
df3 =   1.90854477148950
```

而函数 $y=2^x$ 在 $x=2$ 的二阶导数的准确值 1.92181205567281，从上面的结果可以看出来，第二个公式的精度要相对好一些。对 h 的选择来说，并不是越小越好，从下面的结果就可以看出：

```
>> df = ThreePoint2 ('2^x',2,2,0.001) %步长取 0.001
df =   1.92181213254372
>> df = ThreePoint2 ('2^x',2,2,0.0001) %步长取 0.0001
df =   1.92181204283770
>> df = ThreePoint2 ('2^x',2,2,0.00001) %步长取 0.00001
df =   1.92181381919454
>> df = ThreePoint2 ('2^x',2,2,0.000001) %步长取 0.000001
df =   1.92112992181137
```

精度最好的 h 值为 0.0001，一般情况下都能达到 6 位的精度。

四点公式法求二阶导数的公式如下所示：

$$f''(x_0) \approx \frac{1}{6h^2}(12y_0 - 30y_1 + 24y_2 - 6y_3)$$

$$f''(x_0) \approx \frac{1}{6h^2}(6y_{-1} - 12y_0 + 6y_1)$$

$$f''(x_0) \approx \frac{1}{6h^2}(-6y_{-3} + 24y_{-2} - 30y_{-1} + 12y_0)$$

在 MATLAB 中编程实现的四点公式法求二阶导数的函数为：FourPoint2。

功能：四点公式法求函数的二阶导数。

调用格式：df= FourPoint2 (func,x0,type,h)。

其中，func：函数名；

x0：求导点；

type：采用的四点公式法的形式；

h：步长；

df：二阶导数值。

四点公式法的 MATLAB 程序代码如下所示：

```
function df= FourPoint2(func,x0,type,h)
%采用四点公式法求函数 func 在 x0 处的二阶导数
%函数名： func
%求导点： x0
%四点公式法的形式： type
%步长 ： h
%导数值： df
if nargin == 3
   h = 0.1;
else if (nargin == 4 && h == 0.0)
       disp('h 不能为 0! ');
       return;
   end
end
for i=1:7
    y(i) = subs(sym(func), findsym(sym(func)), x0-3*h+i*h-h);
end
hd = 1/6/h/h;
switch type
   case 1,
       df = (12*y(4) - 30*y(5) + 24*y(6) - 6*y(7))*hd; %用第一个公式求二阶导数
   case 2,
       df = (6*y(3) - 12*y(4) + 6*y(5))*hd;           %用第二个公式求二阶导数
   case 3,
       df = (12*y(4) - 30*y(3) + 24*y(2) - 6*y(1))*hd; %用第三个公式求二阶导数
end
```

例 7-10　四点公式法求二阶导数应用实例。用四点公式法求函数 $y=\sqrt{x}$ 在 $x=4$ 处的二阶导数值。

解： 在 MATLAB 命令窗口中输入：

```
>> df1 = FourPoint2 ('2^x',4,1,0.01)
>> df2 = FourPoint2 ('2^x',4,2,0.01)
>> df3 = FourPoint2 ('2^x',4,3,0.01)
```

输出的计算结果为：

```
df1 =  -0.03124933498633
df2 =   -0.03125006103666
df3 =   -0.03124932214623
```

而函数 $y=\sqrt{x}$ 在 $x=4$ 处的二阶导数值为–0.03125。

五点公式法求二阶导数的公式如下所示：

$$f''(x_0)\approx\frac{1}{12h^2}(35y_0-104y_1+114y_2-56y_3+11y_4)$$

$$f''(x_0)\approx\frac{1}{12h^2}(11y_{-1}-20y_0+6y_1+4y_2-y_3)$$

$$f''(x_0)\approx\frac{1}{12h^2}(-y_{-2}+16y_{-1}-30y_0+16y_1-y_2)$$

$$f''(x_0)\approx\frac{1}{12h^2}(-y_{-3}+4y_{-2}+6y_{-1}-20y_0+11y_1)$$

$$f''(x_0)\approx\frac{1}{12h^2}(11y_{-4}-56y_{-3}+114y_{-2}-104y_{-1}+35y_0)$$

在 MATLAB 中编程实现的五点公式法求二阶导数的函数为：FivePoint2。

功能：五点公式法求函数的二阶导数。

调用格式：df= FivePoint2 (func,x0,type,h)。

其中，func：函数名；

x0：求导点；

type：采用的五点公式法的形式；

h：步长；

df：二阶导数值。

五点公式法的 MATLAB 程序代码如下所示：

```
function df= FivePoint2(func,x0,type,h)
%采用五点公式法求函数 func 在 x0 处的二阶导数
%函数名: func
%求导点: x0
%五点公式法的形式:  type
%步长 : h
%二阶导数值:  df
if nargin == 3
   h = 0.1;
else if (nargin == 4 && h == 0.0)
       disp('h 不能为 0! ');
       return;
   end
end
for i=1:9
    y(i) = subs(sym(func), findsym(sym(func)), x0-4*h+i*h-h);
end
hd = 1/12/h/h;
```

```
switch type
    case 1,
        df = (35*y(5)-104* y(6)+114* y(7)-56* y(8)+11* y(9))*hd;
        %用第一个公式求导数
    case 2,
        df = (11*y(4)-20* y(5)+6* y(6)+4* y(7)- y(8))*hd; %用第二个公式求导数
    case 3,
        df = (-y(3)+16* y(4)-30* y(5)+16* y(6)- y(7))*hd; %用第三个公式求导数
    case 4,
        df = (-y(2)+4*y(3)-6*y(4)+20*y(5)+11*y(6))*hd;     %用第四个公式求导数
    case 5,
        df = (35*y(5)-104* y(4)+114* y(3)-56* y(2)+11* y(1))*hd;
        %用第五个公式求导数
end
```

例 7-11 五点公式法求二阶导数应用实例。用五点公式法求函数 $y=\sqrt{x}$ 在 $x=4$ 处的二阶导数值。

解： 在 MATLAB 命令窗口中输入：

```
>> df1 = FivePoint2 ('2^x',4,1,0.01)
>> df2 = FivePoint2 ('2^x',4,2,0.01)
>> df3 = FivePoint2 ('2^x',4,3,0.01)
>> df4 = FivePoint2 ('2^x',4,4,0.01)
>> df5 = FivePoint2 ('2^x',4,5,0.01)
```

输出的计算结果为：

```
df1 =  -0.03124999476375
df2 =   -0.03125000053172
df3 =   -0.03124999999919
df4 =   -0.03124999946221
df5 =   -0.03125000543669
```

而函数 $y=\sqrt{x}$ 在 $x=4$ 处的二阶导数值为–0.03125。

7.7.2 三次样条法

三次样条也可用来求函数的二阶导数值，而且精度也很高。

用三次样条求已知函数 $f(x)$ 在点 x_0 处的二阶导数的算法介绍如下。

❶ 以 x_0 为中心向两边等分出 n 份，等分间距为 h，即形成下面的分割：

$$x_0-nh<\cdots<x_0-h<x_0<x_0+h<\cdots<x_0+nh$$

❷ 以这些点形成三次样条函数 S，求出 S 的系数 ：

$$S(x)=\sum_{j=-1}^{n+1}c_j\Omega_3\left(\frac{x-x_{-n}}{h}-j\right)$$

其中 $x_{-n}=x_0-nh$。

❸ 按下面的公式求出点 x_0 处的二阶导数：

$$f''(x_0) \approx \frac{6}{h^2}[f(x_0) - c_n]$$

在 MATLAB 中编程实现的三次样条法求二阶导数的函数为：Diff2Bsample。
功能：三次样条法求函数的二阶导数。
调用格式：df= Diff2BSample (func,x0,n,h)。
其中，func：函数名；
　　x0：求导点；
　　n：等分份数；
　　h：步长；
　　df：二阶导数值。

三次样条法求二阶导数的 MATLAB 程序代码如下所示：

```
function df0 = Diff2BSample(func,x0,n,h)
%三次样条法已知函数 func 在点 x0 处的二阶导数
%函数名：func
%求导点： x0
%等分份数： n
%步长：h
%二阶导数值：df
format long;
node_num = 2*n + 1;
for i=1:node_num
    y(i) = subs(sym(func), findsym(sym(func)), x0-n*h+i*h-h);
end
y_1 = (-3*y(1)+4*y(2)-y(3))/(2*h);
y_N = (3*y(2*n+1)-4*y(2*n)+3*y(2*n-1))/(2*h);
c = SubBSample(h,2*n,y,y_1,y_N);
df0 = (y(n+1)-c(n+2))*6/h/h;
format short;
function c = SubBSample(h,n,y,y_1,y_N)
f0 = 0.0;
c = zeros(n+3,1);
b = zeros(n+1,1);
A = diag(4*ones(n+1,1));
I = eye(n+1,n+1);
AL = [I(2:n+1,:);zeros(1,n+1)];
AU = [zeros(1,n+1);I(1:n,:)];
A = A+AL+AU;                    %形成系数矩阵
for i=2:n
   b(i,1) = 6*y(i);
end
b(1) = 6*y(1)+2*h*y_1;
b(n+1) = 6*y(n+1)-2*h*y_N;
d = followup(A,b);              %用追赶法求出系数
c(2:n+2) = d;
c(1) = c(2) - 2*h*y_1;          %c(-1)
```

```
c(n+3) = c(3)+2*h*y_N;           %c(n+1)
```

例 7-12 三次样条法求二阶导数应用实例。用三次样条法求函数 $y=\sqrt{x}$ 在 $x=4$ 处的二阶导数值。

解： 在 MATLAB 命令窗口中输入：

```
>> df0 = Diff2BSample ('sqrt(x)',4,20,0.01)
df0 =   -0.0312
```

而函数 $y=\sqrt{x}$ 在 $x=4$ 处的二阶导数值为–0.03125。

7.8 小结

本章介绍了几种求一阶导数和二阶导数的方法，一般来说，求一阶导数的方法都可稍微修改一下用来求二阶导数。对于高阶导数的求法，可以先用牛顿前插公式和后插公式对原函数进行插值，然后再求其导数，关于此种方法，已有现成的公式可用。

第 8 章　数值积分

数值积分可用于计算解析定义的函数的积分，也可以计算以列表形式给出的函数的积分，其基本原理是采用多项式近似原函数，然后用该多项式的积分近似原函数的积分。

通过本章，读者不仅能掌握常见的数值积分算法，而且还能熟练使用 MATLAB 编程来实现这些算法。

8.1 复合梯形公式法

梯形公式法数值积分采用的梯形公式是最简单的数值积分公式，函数 $f(x)$ 在区间 $[a,b]$ 上的梯形公式法数值积分表达式为：

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{2}[f(a)+f(b)]$$

由于用梯形公式来求积分十分粗糙，误差也比较大，复合梯形公式法积分是将积分区间 $[a,b]$ 划分成 n 个子区间，在每个子区间上应用梯形公式，再求和得到积分结果，复合梯形公式如下：

$$\int_a^b f(x)\mathrm{d}x \approx \frac{h}{2}[f(a)+2\sum_{k=1}^{n-1} f(x_k)+f(b)]$$

其中 $x_k = a+kh$，$h=\dfrac{b-a}{n}$。

采用复合梯形公式法求已知函数 $f(x)$ 的积分的算法步骤如下：

❶ 给定积分区间 $[a,b]$ 和积分精度 eps；

❷ 对 $n=1,2,\cdots$，用复合梯形公式计算

$$I_n(f)=\int_a^b f(x)\mathrm{d}x \approx \frac{h}{2}[f(a)+2\sum_{k=1}^{n-1} f(x_k)+f(b)]$$

if $\left|I_n(f)-I_{n-1}(f)\right| > \text{eps}$，则令 $n=n+1$，继续迭代；

否则结束迭代，退出。

在 MATLAB 中编程实现的复合梯形公式法求积分的函数为：CombineTraprl

功能：复合梯形公式法求积分

调用格式：[q,step]= CombineTraprl(f,a,b,eps)

其中，f：被积函数；

a：积分区间左端点；

b：积分区间右端点；

eps：积分精度；
q：积分值；
step：求得积分所用的子区间数。

复合梯形公式法求积分的 MATLAB 代码如下所示：

```
function [q,step] = CombineTraprl(f,a,b,eps)
%被积函数：f
%积分区间左端点：a
%积分区间右端点：b
%eps  精度
%积分结果：q
%step 积分的子区间数
if(nargin==3)
   eps=1.0e-4;                          %默认精度为 0.0001
end
n=1;
h=(b-a)/2;
q1=0;
q2=(subs(sym(f),findsym(sym(f)),a)+subs(sym(f),findsym(sym(f)),b))/h;
while abs(q2-q1)>eps
   n=n+1;
   h=(b-a)/n;
   q1=q2;
   q2=0;
   for i=0:n-1             %第 n 次的复合梯形公式积分
       x=a+h*i;             %i=0 和 n-1 时，分别代表积分区间的左右端点
       x1=x+h;
       q2=q2+(h/2)*(subs(sym(f),findsym(sym(f)),x)+…
              subs(sym(f),findsym(sym(f)),x1));
   end
end
q=q2;
step=n;
```

例 8-1 复合梯形公式法求数值积分应用实例。计算定积分 $\int_2^4 \frac{1}{x^2-1}\mathrm{d}x$。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> [q,s]=CombineTraprl('1/(x^2-1)',2,4)
q =    0.2945
s =    15
>> [q,s]=CombineTraprl('1/(x^2-1)',2,4,1.0e-6)
q =    0.2939
s =    66
```

所以从复合梯形公式法可得出 $\int_2^4 \frac{1}{x^2-1}\mathrm{d}x \approx 0.2939$。

从第二步结果可以看出，为了达到 $1.0e^{-6}$ 的积分精度，复合梯形公式法采用了 66 个积

分区间。

8.2 辛普森法数值积分

辛普森法数值积分采用辛普森公式来计算，其数值积分公式为：

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{6}[f(a)+4f(\frac{a+b}{2})+f(b)]$$

它的几何意义为用通过三点 $(a,f(a))$、$(\frac{a+b}{2},f(\frac{a+b}{2}))$、$(b,f(b))$ 的抛物线围成的曲边形面积来代替给定函数的积分。精度高一些的还有辛普森 3/8 公式：

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{8}[f(a)+3f(\frac{2a+b}{3})+3f(\frac{a+2b}{3})+f(b)]$$

与梯形公式一样，也有复合辛普森公式：

$$\int_a^b f(x)\mathrm{d}x \approx \frac{h}{6}\sum_{k=0}^{n-1}[f(x_k)+4f(x_{k+\frac{1}{2}})+f(x_{k+1})]$$

$$f(x_0)=f(a), f(x_n)=f(b)$$

其中 $x_{k+\frac{1}{2}}=\frac{x_k+x_{k+1}}{2}$，$h=\frac{b-a}{n}$。

用复合辛普森公式求已知函数 $f(x)$ 的积分的算法步骤如下：

❶ 给定积分区间和积分精度；

❷ 对 $n=1,2,\cdots$，用复合辛普森公式计算

$$I_n(f)=\int_a^b f(x)\mathrm{d}x \approx \frac{h}{6}\sum_{k=0}^{n-1}[f(x_k)+4f(x_{k+\frac{1}{2}})+f(x_{k+1})]$$

if $|I_n(f)-I_{n-1}(f)|>\text{eps}$，则令 $n=n+1$，继续迭代；

否则结束迭代，退出。

在 MATLAB 中编程实现的辛普森系列公式求积分的函数为：IntSimpson

功能：用辛普森系列公式求积分

调用格式：[q,step]= IntSimpson (f,a,b,type,eps)

其中，f：被积函数；

a：积分区间左端点；

b：积分区间右端点；

type：所采用的辛普森公式的类型；

eps：积分精度；

q：积分值；

step：求得积分所用的子区间数。

辛普森系列公式的 MATLAB 代码如下所示：

```
function [q,step] = IntSimpson(f,a,b,type,eps)
```

```
%被积函数: f
%积分区间左端点: a
%积分区间右端点: b
%辛普森公式的类型: type
%eps  精度
%积分结果: q
%积分的子区间数: step
if(type==3 && nargin==4)
   disp('缺少参数! ');
end
q=0;
switch type
   case 1,                %辛普森公式
      q=((b-a)/6)*(subs(sym(f),findsym(sym(f)),a)+…
         4*subs(sym(f),findsym(sym(f)),(a+b)/2)+…
         subs(sym(f),findsym(sym(f)),b));
      step=1;
   case 2,                %辛普森 3/8 公式
      q=((b-a)/8)*(subs(sym(f),findsym(sym(f)),a)+…
         3*subs(sym(f),findsym(sym(f)),(2*a+b)/3)+…
        3*subs(sym(f),findsym(sym(f)),(a+2*b)/3)+subs(sym(f),findsym
(sym(f)),b));
      step=1;
   case 3,                %复合辛普森公式
      n=2;
      h=(b-a)/2;
      q1=0;
q2=(subs(sym(f),findsym(sym(f)),a)+subs(sym(f),findsym(sym(f)),b))/h;
      while abs(q2-q1)>eps
         n=n+1;
         h=(b-a)/n;
         q1=q2;
         q2=0;
         for i=0:n-1
            x=a+h*i;
            x1=x+h;
            q2=q2+(h/6)*(subs(sym(f),findsym(sym(f)),x)+…
               4*subs(sym(f),findsym(sym(f)),(x+x1)/2)+…
               subs(sym(f),findsym(sym(f)),x1));
         end
      end
      q=q2;
      step=n;
end
```

例 8-2 辛普森法数值积分应用实例。计算积分 $\int_0^{10} \sin x dx$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> [q,s]=IntSimpson('sin(x)',0,10,1)
```

```
q =   -7.2995
s =     1
>> [q,s]=IntSimpson('sin(x)',0,10,2)
q =    0.0084
s =     1
>> [q,s]=IntSimpson('sin(x)',0,10,3)
q =    1.8393
s =    13
```

上面的例子是分别用辛普森公式、辛普森 3/8 公式和复合辛普森公式算出来的结果，当然用复合辛普森公式算出来的才是正确的结果，即 $\int_0^{10}\sin x\mathrm{d}x\approx 1.8393$。

函数 sin(x)在区间[0,10]上积分的准确值为–cos(10)+1，大约为 1.8391，由此可见，如果积分区间不是被积函数的单调区间，则辛普森公式和辛普森 3/8 公式的误差将是很大的，而复合辛普森公式显然不存在这种问题。

8.3 牛顿-科茨法数值积分

牛顿-科茨积分公式是通过在 n 个等距节点上用 n–1 阶多项式对被积函数进行插值来逼近被积函数，并对逼近函数求积分来获得。其积分公式的表达式如下所示：

$$\begin{cases}\int_a^b f(x)\mathrm{d}x\approx(b-a)\sum_{k=0}^{n}C_k f(x_k)\\ f(x_0)=f(a)\\ f(x_n)=f(b)\end{cases}$$

梯形公式对应于上式中的 $n=1$ 的情况，辛普森公式对应于上式中的 $n=2$ 的情况。一些比较常用的积分公式如下所示：

- 科茨公式：

$$\int_a^b f(x)\mathrm{d}x\approx\frac{b-a}{90}[7f(a)+32f(\frac{3a+b}{4})+12f(\frac{a+b}{2})+32f(\frac{a+3b}{4})+7f(b)]$$

- 牛顿-科茨六点公式：

$$\int_a^b f(x)\mathrm{d}x\approx\frac{b-a}{288}[19f(a)+75f(\frac{4a+b}{5})+50f(\frac{3a+2b}{5})+50f(\frac{2a+3b}{5})+75f(\frac{a+4b}{5})+19f(b)]$$

- 牛顿-科茨七点公式：

$$\int_a^b f(x)\mathrm{d}x\approx\frac{b-a}{840}[41f(a)+216f(\frac{5a+b}{6})+27f(\frac{2a+b}{3})+272f(\frac{a+b}{2})$$
$$+27f(\frac{a+2b}{3})+216f(\frac{a+5b}{6})+41f(b)]$$

当 $n\leqslant 6$ 时，n 越大，积分精度越高；但是当 $n>6$ 时，积分的稳定性得不到保证，因此一般情况下，n 最大取 6 就够了。

在 MATLAB 中编程实现的牛顿-科茨系列公式求积分的函数为：NewtonCotes
功能：用牛顿-科茨系列公式求积分

调用格式：q= NewtonCotes (f,a,b,type)
其中，f：被积函数；
a：积分区间左端点；
b：积分区间右端点；
type：所采用的牛顿-科茨系列公式的类型；
q：积分值。

牛顿-科茨系列公式的 MATLAB 代码如下所示：

```
function q = NewtonCotes(f,a,b,type)
%被积函数：f
%积分区间左端点：a
%积分区间右端点：b
%牛顿－科茨公式的类型：type
%积分结果：q
if(type==3 && nargin==4)
   eps=1.0e-4;                          %默认精度为 0.0001
end
q=0;
switch type
   case 1,                              %科茨公式
      q=((b-a)/90)*(7*subs(sym(f),findsym(sym(f)),a)+…
         32*subs(sym(f),findsym(sym(f)),(3*a+b)/4)+…
         12*subs(sym(f),findsym(sym(f)),(a+b)/2)+…
         32*subs(sym(f),findsym(sym(f)),(a+3*b)/4)+…
          7*subs(sym(f),findsym(sym(f)),b));
   case 2,                    %牛顿-科茨六点公式
      q=((b-a)/288)*(19*subs(sym(f),findsym(sym(f)),a)+…
         75*subs(sym(f),findsym(sym(f)),(4*a+b)/5)+…
         50*subs(sym(f),findsym(sym(f)),(3*a+2*b)/5)+…
         50*subs(sym(f),findsym(sym(f)),(2*a+3*b)/5)+…
         75*subs(sym(f),findsym(sym(f)),(a+4*b)/5)+…
         19*subs(sym(f),findsym(sym(f)),b));
   case 3,                              %牛顿-科茨七点公式
      q=((b-a)/840)*(41*subs(sym(f),findsym(sym(f)),a)+…
         216*subs(sym(f),findsym(sym(f)),(5*a+b)/6)+…
         27*subs(sym(f),findsym(sym(f)),(2*a+b)/3)+…
         272*subs(sym(f),findsym(sym(f)),(a+b)/2)+…
         27*subs(sym(f),findsym(sym(f)),(a+2*b)/3)+…
         216*subs(sym(f),findsym(sym(f)),(a+5*b)/6)+…
         41*subs(sym(f),findsym(sym(f)),b));
end
```

例 8-3 牛顿-科茨系列公式数值积分应用实例。计算积分 $\int_0^{10}\sin x\mathrm{d}x$。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> q=NewtonCotes('sin(x)',0,10,1)
```

```
q =    3.7613
>> q=NewtonCotes('sin(x)',0,10,2)
q =    2.7865
>> q=NewtonCotes('sin(x)',0,10,3)
q =    1.5296
```

上面的例子是分别用科茨公式、牛顿-科茨六点公式和牛顿-科茨公式七点公式算出来的结果，当然第三个结果才是准确的，即 $\int_0^{10} \sin x \mathrm{d}x \approx 1.5296$。这个例子说明低阶的牛顿-科茨公式也存在同辛普森公式一样的缺点。

8.4 高斯系列公式数值积分

8.4.1 高斯公式

高斯积分公式的思想是用 n 个不等距的节点 $x_1, x_2, \cdots, x_n$ 对被积函数进行插值，然后对插值后的函数进行积分，其积分公式为：

$$\int_{-1}^{1} f(x)\mathrm{d}x \approx \sum_{k=1}^{n} A_k f(x_k)$$

如果积分区间不是 $[-1,1]$，则需转换到此区间：

$$\int_a^b f(x)\mathrm{d}x = \frac{b-a}{2}\int_{-1}^{1} f(\frac{b-a}{2}t + \frac{a+b}{2})\mathrm{d}t$$

其中系数 A_k、节点 x_k 与 n 的关系如表 8-1 所示：

表 8-1 高斯公式节点和系数表

n	x_k	A_k
1	0.0000000	2.0000000
2	±0.5773503	1.0000000
3	±0.7745967	0.55555556
	0.0000000	0.88888889
4	±0.8611363	0.3478548
	±0.3398810	0.6521452
5	±0.9061793	0.2369269
	±0.5384693	0.4786287
	0.0000000	0.5688889

在 MATLAB 中编程实现的高斯公式求积分的函数为：IntGauss

功能：用高斯公式求积分

调用格式：q= IntGauss(f,a,b,n,AK,XK)

其中，f：被积函数；

a：积分区间左端点；

b：积分区间右端点；

n：所采用的高斯积分点个数；
AK：自定义系数
XK：自定义积分点；
q：积分值。

高斯公式的 MATLAB 代码如下所示：

```
function q = IntGauss(f,a,b,n,AK,XK)
%被积函数：f;
%积分区间左端点：a;
%积分区间右端点：b;
%所采用的高斯积分点个数：n;
%自定义系数：AK
%自定义积分点：XK;
%积分值：q;
if(n<5 && nargin == 4)
    AK = 0;
    XK = 0;
Else                                          %如果 n>4，则节点和系数由调用者给出
    XK1=((b-a)/2)*XK+((a+b)/2);
    q=((b-a)/2)*sum(AK.*subs(sym(f),findsym(f),XK1));
end
ta = (b-a)/2;
tb = (a+b)/2;
switch n
    case 1, %n=1 时，采用表 8-1 中对应的系数 xk 和 Ak 进行计算
        q=2*ta*subs(sym(f),findsym(sym(f)),tb);
    case 2, %n=2 时，采用表 8-1 中对应的系数 xk 和 Ak 进行计算
        q=ta*(subs(sym(f),findsym(sym(f)),ta*0.5773503+tb)+…
            subs(sym(f),findsym(sym(f)),-ta*0.5773503+tb));
    case 3, %n=3 时，采用表 8-1 中对应的系数 xk 和 Ak 进行计算
        q=ta*(0.55555556*subs(sym(f),findsym(sym(f)),ta*0.7745967+tb)+…
            0.55555556*subs(sym(f),findsym(sym(f)),-ta*0.7745967+tb)+…
            0.88888889*subs(sym(f),findsym(sym(f)),tb));
    case 4, %n=4 时，采用表 8-1 中对应的系数 xk 和 Ak 进行计算
        q=ta*(0.3478548*subs(sym(f),findsym(sym(f)),ta*0.8611363+tb)+…
            0.3478548*subs(sym(f),findsym(sym(f)),-ta*0.8611363+tb)+…
            0.6521452*subs(sym(f),findsym(sym(f)),ta*0.3398810+tb)…
            +0.6521452*subs(sym(f),findsym(sym(f)),-ta*0.3398810+tb));
   case 5,  %n=5 时，采用表 8-1 中对应的系数 xk 和 Ak 进行计算
        q=ta*(0.2369269*subs(sym(f),findsym(sym(f)),ta*0.9061793+tb)+…
            0.2369269*subs(sym(f),findsym(sym(f)),-ta*0.9061793+tb)+…
            0.4786287*subs(sym(f),findsym(sym(f)),ta*0.5384693+tb)…
            +0.4786287*subs(sym(f),findsym(sym(f)),-ta*0.5384693+tb)+…
            0.5688889*subs(sym(f),findsym(sym(f)),tb));
end
```

例 8-4 高斯公式数值积分应用实例 1。计算积分 $\int_0^1 x^9 \mathrm{d}x$。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> q=IntGauss('x^9',0,1,4)
q =  0.0999
```

所以由高斯公式可得到 $\int_0^1 x^9 \mathrm{d}x \approx 0.0999$。

可以证明高斯公式对多项式积分是十分精确的。

例 8-5 高斯公式数值积分应用实例 2。计算积分 $\int_0^{10} \sin x \mathrm{d}x$。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> q=IntGauss('sin(x)',0,10,5)
q =  1.8167
```

所以由高斯公式可得到 $\int_0^{10} \sin x \mathrm{d}x \approx 1.8167$。

从例 8-5 可以看出，高斯公式对于被积函数在积分区间上不单调的情况下不存在同辛普森公式的缺点。

8.4.2 高斯-拉道公式

高斯-拉道求积公式如下：

$$I(f) = \int_{-1}^{1} f(x)\mathrm{d}x \approx \frac{2}{n^2} f(-1) + \sum_{k=2}^{n} A_k f(x_k)$$

前几个高斯-拉道节点和系数的取值与 n 的关系如表 8-2 所示：

表 8-2 高斯-拉道公式节点和系数表

n	x_k	A_k
2	−1	0.5
	0.333333	1.5
3	−1	0.222222
	−0.289898	0.752806
	0.689898	1.024972
4	−1	0.125000
	−0.575319	0.657689
	0.181066	0.776387
	0.822824	0.440925
5	−1	0.080000
	−0.720480	0.446207
	−0.167181	0.623653
	0.446314	0.562712
	0.885792	0.287427

在 MATLAB 中编程实现的高斯-拉道公式求积分的函数为：IntGaussLada

功能：用高斯-拉道公式求积分

调用格式：q= IntGaussLada(f,a,b,n,AK,XK)

其中，f：被积函数；

a：积分区间左端点；

b：积分区间右端点；

n：所采用的积分点个数；

AK：自定义系数；

XK：自定义积分点；

q：积分值。

高斯-拉道公式的 MATLAB 代码如下所示：

```
function q = IntGaussLada(f,a,b,n,AK,XK)
%被积函数：f;
%积分区间左端点：a;
%积分区间右端点：b;
%所采用的积分点个数：n;
%自定义系数：AK
%自定义积分点：XK;
%积分值：q;
if(n<6 && nargin == 4)
   AK = 0;
   XK = 0;
else                              %如果 n>5，则节点和系数由调用者给出
   XK1=((b-a)/2)*XK+((a+b)/2);
   q=((b-a)/2)*((2/n/n)*subs(sym(f),findsym(sym(f)),a)+…
            sum(AK.*subs(sym(f),findsym(sym(f)),XK1)));
end
ta = (b-a)/2;
tb = (a+b)/2;
switch n              %以下是对不同的 n 采取不同的结点和系数
   case 2,          %n = 2 时，采用表 8-2 中对应的系数进行计算
      q=ta*0.5*subs(sym(f),findsym(sym(f)),ta*(-1)+tb)+…
         1.5*ta*subs(sym(f),findsym(sym(f)),ta*(1/3)+tb);
   case 3,          %n = 3 时，采用表 8-2 中对应的系数进行计算
      q=ta*((2/9)*subs(sym(f),findsym(sym(f)),ta*(-1)+tb)+…
         0.752806*subs(sym(f),findsym(sym(f)),ta*(-0.289898)+tb)+…
         1.024972*subs(sym(f),findsym(sym(f)),ta*0.689898+tb));
   case 4,          %n = 4 时，采用表 8-2 中对应的系数进行计算
      q=ta*(0.125*subs(sym(f),findsym(sym(f)),ta*(-1)+tb)+...
         0.657689*subs(sym(f),findsym(sym(f)),-ta*0.575319+tb)+…
         0.776387*subs(sym(f),findsym(sym(f)),ta*0.181066+tb)+…
         0.440925*subs(sym(f),findsym(sym(f)),ta*0.822824+tb));
   case 5,        %n = 5 时，采用表 8-2 中对应的系数进行计算
      q=ta*(0.08*subs(sym(f),findsym(sym(f)),ta*(-1)+tb)+…
```

```
            0.446207*subs(sym(f),findsym(sym(f)),-ta*0.720480+tb)+…
            0.623653*subs(sym(f),findsym(sym(f)),-ta*0.167181+tb)+…
            0.562712*subs(sym(f),findsym(sym(f)),ta*0.446314+tb)+…
            0.287427*subs(sym(f),findsym(sym(f)),ta*0.885792+tb));
end
```

例 8-6 高斯-拉道公式数值积分应用实例。计算积分 $\int_0^{10} e^{-x}\sin x dx$ 。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> q=IntGaussLada('exp(-x)*sin(x)',0,10,3)
q =  -0.0421
>> q=IntGaussLada('exp(-x)*sin(x)',0,10,4)
q =  0.3310
>> q=IntGaussLada('exp(-x)*sin(x)',0,10,5)
q =  0.5035
```

所以由高斯-拉道公式可得到 $\int_0^{10} e^{-x}\sin x dx \approx 0.5035$ 。

此积分的准确值为 0.5000，因此随着 n 的增大，高斯系列公式的精度是越来越高的。

8.4.3 高斯-洛巴托公式

高斯-洛巴托求积公式如下：

$$I(f)=\int_{-1}^{1} f(x)dx \approx \frac{2}{n(n-1)}[f(-1)+f(1)]+\sum_{k=2}^{n-1} A_k f(x_k)$$

高斯-洛巴托公式的前几个节点和系数取值见表 8-3。

表 8-3 高斯-洛巴托公式节点和系数表

n	x_k	A_k
3	0	1.333333
	±1	0.333333
4	±0.447214	0.833333
	±1	0.166666
5	0	0.711111
	±0.654654	0.544444
	±1	0.100000
6	±0.285232	0.554858
	±0.765055	0.378475
	±1	0.066667

在 MATLAB 中编程实现的高斯-洛巴托公式求积分的函数为：IntGaussLobato

功能：用高斯-洛巴托公式求积分

调用格式：q= IntGaussLobato (f,a,b,n,AK,XK)

其中，f：被积函数；

a：积分区间左端点；
b：积分区间右端点；
n：所采用的积分点个数；
AK：自定义系数；
XK：自定义积分点；
q：积分值。

高斯-洛巴托公式的 MATLAB 代码如下所示：

```
function q = IntGaussLobato(f,a,b,n,AK,XK)
%被积函数：f；
%积分区间左端点：a；
%积分区间右端点：b；
%所采用的积分点个数：n；
%自定义系数：AK
%自定义积分点：XK；
%积分值：q；
if(n<7 && nargin == 4)
    AK = 0;
    XK = 0;
else                                            %如果 n>6，则节点和系数由调用者给出
    XK1=((b-a)/2)*XK+((a+b)/2);
    q=((b-a)/2)*((2/n/(n-1))*(subs(sym(f),findsym(sym(f)),a)+…
           subs(sym(f),findsym(sym(f)),b)+…
            sum(AK.*subs(sym(f),findsym(sym(f)),XK1)));
end
ta = (b-a)/2;
tb = (a+b)/2;
switch n                                     %以下是对不同的 n 采取不同的结点和系数
    case 3,   %n=3 时，采用表 8-3 中对应的系数进行计算
       q=ta*((1/3)*(subs(sym(f),findsym(sym(f)),a)+…
           subs(sym(f),findsym(sym(f)),b)+…
           1.333333*subs(sym(f),findsym(sym(f)),tb));
    case 4,    %n=4 时，采用表 8-3 中对应的系数进行计算
       q=ta*((1/6)*(subs(sym(f),findsym(sym(f)),a)+…
           subs(sym(f),findsym(sym(f)),b))+0.833333*…
           (subs(sym(f),findsym(sym(f)),ta*0.447214+tb)+…
           subs(sym(f),findsym(sym(f)),-ta*0.447214+tb)));
    case 5,    %n=5，采用表 8-3 中对应的系数进行计算
        q=ta*((1/10)*(subs(sym(f),findsym(sym(f)),a)+…
            subs(sym(f),findsym(sym(f)),b))+0.544444*…
           (subs(sym(f),findsym(sym(f)),ta*0.654654+tb)+…
           subs(sym(f),findsym(sym(f)),-ta*0.654654+tb))+ …
           0.711111*subs(sym(f),findsym(sym(f)),tb));
    case 6,    %n=6，采用表 8-3 中对应的系数进行计算
        q=ta*((1/15)*(subs(sym(f),findsym(sym(f)),a)+…
            subs(sym(f),findsym(sym(f)),b))+0.554858*…
           (subs(sym(f),findsym(sym(f)),ta*0.285232+tb)+…
```

```
            subs(sym(f),findsym(sym(f)),-ta*0.285232+tb))+…
            0.378475*(subs(sym(f),findsym(sym(f)),ta*0.765055+tb)+…
            subs(sym(f),findsym(sym(f)),-ta*0.765055+tb)));
end
```

例 8-7 高斯-洛巴托公式数值积分应用实例。计算积分 $\int_0^{10} e^{-x}\sin x dx$。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> q=IntGaussLobato('exp(-x)*sin(x)',0,10,4)
q =  0.0993
>> q=IntGaussLobato('exp(-x)*sin(x)',0,10,5)
q =  0.4560
>> q=IntGaussLobato('exp(-x)*sin(x)',0,10,6)
q =  0.5075
```

所以由高斯-洛巴托公式可得到 $\int_0^{10} e^{-x}\sin x dx \approx 0.5075$。

上面求出的值可以和积分的准确值 0.5000 对照比较。如果积分节点个数为 n 的话，高斯公式的积分精度为 $2n-1$；高斯-拉道公式的积分精度为 $2n-2$；高斯-洛巴托公式的积分精度为 $2n-3$。但是如果知道被积函数在区间的端点上取值为 0 的话，宜采用高斯-拉道公式或高斯-洛巴托公式。

8.5 区间逐次分半法数值积分

区间逐次分半法本质上是一种复合积分法，它通过把积分区间逐次分半，以达到想要的积分精度。下面讲述常见的三种区间逐次分半法积分，分别是区间逐次分半梯形公式数值积分、区间逐次分半辛普森数值积分和区间逐次分半布尔数值积分。

8.5.1 区间逐次分半梯形公式数值积分

区间逐次分半梯形公式为：

$$\int_a^b f(x)dx \approx \frac{h}{2}\sum_{k=1}^{2^n}[f(x_{k-1})+f(x_k)]$$

$$f(x_0)=f(a), f(x_{2^n})=f(b)$$

$$h=\frac{b-a}{2^n}$$

在 MATLAB 中编程实现的区间逐次分半梯形公式求积分的函数为：DDTraprl

功能：用区间逐次分半梯形公式求积分

调用格式：[q,step] = DDTraprl(f,a,b,eps)

其中，f：被积函数；

a：积分区间左端点；

b：积分区间右端点；

eps：精度要求；
q：积分结果；
step：积分的子区间数。

区间逐次分半梯形公式的 MATLAB 代码如下所示：

```
function [q,step] = DDTraprl(f,a,b,eps)
%被积函数：f
%积分区间左端点：a
%积分积分区间右端点：b
%精度：eps
%积分结果：q
%积分的子区间数：step
if(nargin==3)
   eps=1.0e-4;
end;
n=1;                              %n 为划分子区间的次数，也就是公式中的 n
h=(b-a);
q2=(subs(sym(f),findsym(sym(f)),a)+subs(sym(f),findsym(sym(f)),b))/h/2;
Tol=1;
while tol>eps
   n=n+1;
   h=h/2;                         %区间逐次分半
   q1=q2;
   q2=0;
   for i=0:(2^n-1)
      x=a+h*i;
      x1=x+h;
      q2=q2+(h/2)*(subs(sym(f),findsym(sym(f)),x)+…
         subs(sym(f),findsym(sym(f)),x1));        %梯形公式
   end
   tol=abs(q2-q1);
end
q=q2;
step=n;
```

例 8-8 区间逐次分半梯形公式数值积分应用实例 1。用区间逐次分半梯形公式计算积分 $\int_0^{10} \mathrm{e}^{-x}\sin x\mathrm{d}x$。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> [q,s]=DDTraprl('exp(-x)*sin(x)',0,10,4)
q =    0.5000
s =    10
```

所以由区间逐次分半梯形公式可得到 $\int_0^{10} \mathrm{e}^{-x}\sin x\mathrm{d}x \approx 0.5000$。

例 8-9 区间逐次分半梯形公式数值积分应用实例 2。用区间逐次分半梯形公式计算积分 $\int_0^{10} \log(x^2+1)\mathrm{d}x$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> [q,s]=DDTraprl('log(x^2+1)',0,10)
q =  29.0935
s =  9
```

可见，由区间逐次分半梯形公式可得到 $\int_0^{10}\log(x^2+1)\mathrm{d}x \approx 29.0935$。

8.5.2 区间逐次分半辛普森公式数值积分

区间逐次分半辛普森公式：

$$\int_a^b f(x)\mathrm{d}x \approx \frac{h}{3}\sum_{k=1}^{2^{n-1}}[f(x_{2k-2})+4f(x_{2k-1})+f(x_{2k})]$$

$$f(x_0)=f(a), f(x_{2^n})=f(b)$$

$$h=\frac{b-a}{2^n}$$

在 MATLAB 中编程实现的区间逐次分半辛普森公式求积分的函数为：DDSimpson

功能：用区间逐次分半辛普森公式求积分

调用格式：[q,step] = DDSimpson (f,a,b,eps)

其中，f：被积函数；

a：积分区间左端点；

b：积分区间右端点；

eps：精度要求；

q：积分结果；

step：积分的子区间数。

区间逐次分半辛普森公式的 MATLAB 代码如下所示：

```
function [q,step] = DDSimpson(f,a,b,eps)
%被积函数：f
%积分区间左端点：a
%积分区间右端点：b
%精度：eps
%积分结果：q
%积分的子区间数：step
if(nargin==3)
    eps=1.0e-4;
end;
n=1;                                    %n 为划分子区间的次数，也就是公式中的 n
h=b-a;
q1=0;
q2=(subs(sym(f),findsym(sym(f)),a)+…
    subs(sym(f),findsym(sym(f)),b))/h/2;
tol=1;
while tol>eps
```

```
        n=n+1;
        h=h/2;                                  %区间逐次分半
        q1=q2;
        q2=0;
        for i=0:(2^(n-1)-1)
            x=a+h*2*i;
            x1=x+h;
            x2=x1+h;
            q2=q2+(h/3)*(subs(sym(f),findsym(sym(f)),x)+…
                4*subs(sym(f),findsym(sym(f)),x1)+…
                subs(sym(f),findsym(sym(f)),x2));          %辛普森公式
        end
        tol=abs(q2-q1);
    end
    q=q2;
    step=n;
```

例 8-10 区间逐次分半辛普森公式数值积分应用实例。计算积分 $\int_0^{10} \mathrm{e}^{-x}\sin x\mathrm{d}x$。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> [q,s]=DDSimpson('exp(-x)*sin(x)',0,10)
q =  0.5000
s =  8
```

所以由区间逐次分半辛普森公式可得到 $\int_0^{10} \mathrm{e}^{-x}\sin x\mathrm{d}x \approx 0.5000$。

对于同样的积分精度，区间逐次分半辛普森公式比区间逐次分半梯形公式划分的子区间次数要少。

8.5.3 区间逐次分半布尔公式数值积分

区间逐次分半布尔公式为：

$$\begin{cases} \int_a^b f(x)\mathrm{d}x \approx \dfrac{2h}{45}\sum_{k=1}^{4^{n-1}}[7f(x_{4k-4})+32f(x_{4k-3})+12f(x_{4k-2})+32f(x_{4k-1})+7f(x_{4k})] \\ f(x_0)=f(a), f(x_{4^n})=f(b) \\ h=\dfrac{b-a}{4^n} \end{cases}$$

在 MATLAB 中编程实现的区间逐次分半布尔公式求积分的函数为：DDBuer

功能：用区间逐次分半布尔公式求积分

调用格式：[q,step] = DDBuer (f,a,b,eps)

其中，f：被积函数；

a：积分区间左端点；

b：积分区间右端点；

eps：精度要求；

q：积分结果；
step：积分的子区间数。

区间逐次分半布尔公式的 MATLAB 代码如下所示：

```
function [q,step] = DDBuer(f,a,b,eps)
%被积函数：f
%积分区间左端点：a
%积分区间右端点：b
%精度：eps
%积分结果：q
%积分的子区间数：step
if(nargin==3)
   eps=1.0e-4;
end;
n=1;                              %n 为划分子区间的次数，也就是公式中的 n
h=b-a;
q1=0;
q2=(subs(sym(f),findsym(sym(f)),a)+subs(sym(f),findsym(sym(f)),b))/h/2;
tol=1;
while tol>eps
   n=n+1;
   h=h/4;                         %注意是除以 4!
   q1=q2;
   q2=0;
   for i=0:(4^(n-1)-1)
       x=a+h*4*i;
       x1=x+h;
       x2=x1+h;
       x3=x2+h;
       x4=x3+h;
       q2=q2+(2*h/45)*(7*subs(sym(f),findsym(sym(f)),x)+…
           32*subs(sym(f),findsym(sym(f)),x1)+…
           12*subs(sym(f),findsym(sym(f)),x2)+…
           32*subs(sym(f),findsym(sym(f)),x3)+…
           7*subs(sym(f),findsym(sym(f)),x4));              %布尔公式
   end
   tol=abs(q2-q1);
end
q=q2;
step=n;
```

例 8-11 区间逐次分半布尔公式数值积分应用实例。计算积分 $\int_0^{10} e^{-x}\sin x dx$。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> [q,s]=DDBuer('exp(-x)*sin(x)',0,10)
q =    0.5000
s =    5
```

所以由区间逐次分半布尔公式可得到$\int_0^{10} e^{-x}\sin x dx \approx 0.5000$。

对于同样的积分精度，区间逐次分半布尔公式的速度是最快的。

8.6 龙贝格积分法

龙贝格积分法是用里查森外推算法来加快复合梯形求积公式的收敛速度，它的算法如下，其中$T_m^{(i)}$是一系列逼近原定积分的龙贝格积分值。

❶ 计算

$$T_1^{(0)}=\frac{b-a}{2}[f(a)+f(b)]$$

❷ 对$k=1,2,3,\cdots n$，计算下列各步：

$$T_1^{(k)}=\frac{1}{2}[T_1^{(k-1)}+\frac{b-a}{2^{k-1}}\sum_{j=1}^{2k-1}f(a+\frac{(2j-1)(b-a)}{2^k})]$$

对$m=1,2,\cdots,k$和$i=k,k-1,k-2,\cdots,1$，计算

$$T_{m+1}^{i-1}=\frac{4^m T_m^i-T_m^{i-1}}{4^m-1}$$

❸ 精度控制

把上面的计算过程用表 8-4 表示如下。

表 8-4　龙贝格积分计算表格

$T_1^{(0)}$			
$T_1^{(1)}$	$T_2^{(0)}$		
$T_1^{(2)}$	$T_2^{(1)}$	$T_3^{(0)}$	
$T_1^{(3)}$	$T_2^{(2)}$	$T_3^{(1)}$	
$T_1^{(4)}$	$T_2^{(3)}$	$T_3^{(2)}$	
⋮	⋮	⋮	…

随着计算的步骤的增加，$T_m^{(i)}$越来越逼近积分$\int_a^b f(x)dx$。下面的代码是用$T_m^{(m)}$来逼近$\int_a^b f(x)dx$的 MATLAB 代码。

在 MATLAB 中编程实现的龙贝格公式求积分的函数为：Roberg
功能：用龙贝格公式求积分
调用格式：[q,step] = Roberg (f,a,b,eps)
其中，f：被积函数；
　　a：积分区间左端点；
　　b：积分区间右端点；
　　eps：精度要求；
　　q：积分结果；

step：积分的子区间数。

龙贝格积分的 MATLAB 代码如下所示：

```
function [q,step]=Roberg(f,a,b,eps)
%被积函数：f
%积分区间左端点：a
%积分区间右端点：b
%精度：eps
%积分结果：q
%积分的子区间数：step
if(nargin==3)
    eps=1.0e-4;
end;
M=1;
tol=10;
k=0;
T=zeros(1,1);
h=b-a;
T(1,1)=(h/2)*(subs(sym(f),findsym(sym(f)),a)+subs(sym(f),findsym(sym(f)
),b));
                                    %初始值
while tol>eps
    k=k+1;
    h=h/2;
    Q=0;
    for i=1:M
        x=a+h*(2*i-1);
        Q=Q+subs(sym(f),findsym(sym(f)),x);
    end
    T(k+1,1)=T(k,1)/2+h*Q;     %龙贝格积分表中的第 k 行第一列的积分值
    M=2*M;
    for j=1:k
        T(k+1,j+1)=T(k+1,j)+(T(k+1,j)-T(k,j))/(4^j-1);
        %龙贝格积分表中的第 k 行其余积分值
    end
    tol=abs(T(k+1,j+1)-T(k,j));  %精度控制
end
q=T(k+1,k+1);
step=k;
```

例 8-12 龙贝格公式数值积分应用实例 1。计算积分 $\int_{-1}^{1} x^2 dx$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> [q,s]=Roberg('x^2',-1,1)
q =    0.6667
s =    2
```

所以由龙贝格公式可得到 $\int_{-1}^{1} x^2 dx \approx 0.6667$。

例 8-13 龙贝格公式数值积分应用实例 2。计算积分 $\int_0^1 \frac{\sin x}{\sin x + \cos x} \mathrm{d}x$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> [q,s]=Roberg('sin(x)/(sin(x)+cos(x))',0,1)
q =  0.3383
s =  3
```

所以由龙贝格公式可得到 $\int_0^1 \frac{\sin x}{\sin x + \cos x} \mathrm{d}x \approx 0.3383$。

8.7 自适应法求积分

自适应积分法是一种比较经济而且快速的求积分的方法。它能自动地在被积函数变化剧烈的地方增多节点，而在被积函数变化平缓的地方减少节点。因此它是一种不均匀区间的积分方法。按照子区间上的积分方式它可以分为自适应辛普森积分法和自适应梯形积分法。通常采用前者作为子区间的积分方式。

自适应积分法的基本步骤如下：

❶ 将积分区间 $[a,b]$ 分成两个相等的 1 级子区间 $[a,a+\frac{h}{2}]$ 和 $[a+\frac{h}{2},a+h]$，且 $h=b-a$；

❷ 在这上述两个 1 级子区间上用辛普森积分得到积分 $I^{(1)}_{a,a+\frac{h}{2}}$ 和 $I^{(1)}_{a+\frac{h}{2},a+h}$；

❸ 将子区间 $[a,a+\frac{h}{2}]$ 分成两个相等的 2 级子区间 $[a,a+\frac{h}{2^2}]$ 和 $[a+\frac{h}{2^2},a+\frac{h}{2}]$；

❹ 采用辛普森积分计算得到：

$$I^{(2)}_{a,a+\frac{h}{2}} = I^{(1)}_{a,a+\frac{h}{2^2}} + I^{(1)}_{a+\frac{h}{2^2},a+\frac{h}{2}}$$

❺ 比较 $I^{(2)}_{a,a+\frac{h}{2}}$ 和 $I^{(1)}_{a,a+\frac{h}{2}}$，如果 $\left|I^{(1)}_{a,a+\frac{h}{2}} - I^{(2)}_{a,a+\frac{h}{2}}\right| < 10\times\frac{\varepsilon}{2}$，其中 ε 为整体积分所需精度，则认为子区间 $[a,a+\frac{h}{2}]$ 上的积分 $I^{(1)}_{a,a+\frac{h}{2}}$ 已达到所需精度，不需要再细分；否则就需要再细分，对每个 2 级子区间做同样的判断。

1 级子区间 $[a+\frac{h}{2},a+h]$ 的操作过程与上面完全相同。

在 MATLAB 中编程实现的自适应辛普森求积分的函数为：SmartSimpson
功能：用自适应辛普森公式求积分
调用格式：q = SmartSimpson (f,a,b,eps)
其中，f：被积函数；
　　a：积分区间左端点；

b：积分区间右端点；
eps：精度要求；
q：积分结果。

自适应辛普森积分的 MATLAB 代码如下所示：

```
function q=SmartSimpson(f,a,b,eps)
%被积函数：f
%积分区间左端点：a
%积分区间右端点：b
%精度：eps
%积分结果：q
if(nargin==3)
   eps=1.0e-4;
end;
e=5*eps;
q=SubSmartSimpson(f,a,b,e);
function q=SubSmartSimpson(f,a,b,eps)
QA=IntSimpson(f,a,b,1,eps);                  %对整个区间用辛普森积分
QLeft=IntSimpson(f,a,(a+b)/2,1,eps);         %对左子区间用辛普森积分
QRight=IntSimpson(f,(a+b)/2,b,1,eps);        %对右子区间用辛普森积分
if(abs(QLeft+QRight-QA)<=eps)
   q=QA;          %精度足够则返回积分值
else
   q=SubSmrtSimpson(f,a,(a+b)/2,eps)+SubSmartSimpson(f,(a+b)/2,b,eps);
%递归公式
end
```

例 8-14 自适应辛普森积分公式数值积分应用实例 1。计算积分 $\int_0^1 x\sin x\mathrm{d}x$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> q=SmartSimpson('x*sin(x)',0,1)
q =  0.3011
```

所以由自适应辛普森积分公式可得到 $\int_0^1 x\sin x\mathrm{d}x \approx 0.3011$。

例 8-15 自适应辛普森积分公式数值积分应用实例 2。计算积分 $\int_1^2 \frac{1}{\sin x+\sqrt{x}}\mathrm{d}x$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> q=SmartSimpson('1/(sqrt(x)+sin(x))',1,2)
q = 0.4622
```

所以由自适应辛普森积分公式可得到 $\int_1^2 \frac{1}{\sin x+\sqrt{x}}\mathrm{d}x \approx 0.4622$。

自适应辛普森积分应用十分广泛，不管被积函数多复杂，它都能快速地得到高精度的结果，MATLAB 中的积分函数 quad 就采用自适应辛普森积分方法。

8.8 三次样条函数求积分

用三次样条求积分 $I(f)=\int_a^b f(x)\mathrm{d}x$ 的算法介绍如下。

❶ 将区间 $[a,b]$ 等分成 n 等份，节点为：

$$x_k = a + kh \qquad (k = 0,1,2,\cdots,n)$$

其中 $b = a + nh$；

并在两端点各延拓一点：

$$x_{-1} = a - h$$
$$x_{n+1} = a + (n+1)h$$

❷ 以这些节点形成三次样条函数 S，求出 S 的系数：

$$S(x) = \sum_{j=-1}^{n+1} c_j \Omega_3(\frac{x - x_k}{h})\text{；}$$

其中 $\Omega_3(\frac{x - x_k}{h})$ 为 B 样条函数；

❸ 按下面的公式求出积分：

$$I(f) \approx \int_a^b S(x)\mathrm{d}x = \frac{h}{24}(c_{-1} + c_{n+1}) + \frac{h}{2}(c_0 + c_n) + \frac{23}{24}h(c_1 + c_{n-1}) + h\sum_{k=2}^{n-2} c_k$$

在 MATLAB 中编程实现的三次样条求积分的函数为：IntSample

功能：用三次样条插值求积分

调用格式：q= IntSample(func,a,b,n)

其中，func：被积函数；

a：积分区间左端点；

b：积分区间右端点；

n：积分区间等份数；

q：积分值。

三次样条求积分的 MATLAB 代码如下所示：

```
function q = IntSample(func,a,b,n)
%被积函数：func
%积分区间左端点：a
%积分区间右端点：b
%积分区间等份数：n
%积分结果：q
format long;
node_num = n + 3;         %加上延拓的 2 个节点共 n+3 个节点
h = (b-a)/n;
```

```
    for i=1:node_num
        y(i) = subs(sym(func), findsym(sym(func)), a+i*h-2*h);
    end
    y_1 = (-3*y(1)+4*y(2)-y(3))/(2*h);
    y_N = (3*y(node_num)-4*y(node_num-1)+3*y(node_num-2))/(2*h);
    c = SubBSample(h,node_num-1,y,y_1,y_N);  %用样条逼近法求出样条系数
    q = h*(c(1)+c(node_num+2))/24+h*(c(2)+c(node_num+1))/2+23*h*(c(3)+
c(node_num))/24;;
    for j=4:node_num-1
       q = q+h*c(j);
    end
    format short;
    function c = SubBSample(h,n,y,y_1,y_N)
    f0 = 0.0;
    c = zeros(n+3,1);
    b = zeros(n+1,1);
    A = diag(4*ones(n+1,1));
    I = eye(n+1,n+1);
    AL = [I(2:n+1,:);zeros(1,n+1)];
    AU = [zeros(1,n+1);I(1:n,:)];
    A = A+AL+AU;                          %形成系数矩阵
    for i=2:n
       b(i,1) = 6*y(i);
    end
    b(1) = 6*y(1)+2*h*y_1;
    b(n+1) = 6*y(n+1)-2*h*y_N;
    d = followup(A,b);                    %用追赶法求出系数
    c(2:n+2) = d;
    c(1) = c(2) - 2*h*y_1;                %c(-1)
    c(n+3) = c(3)+2*h*y_N;                %c(n+1)
```

例 8-16 三次样条函数求积分应用实例。采用样条函数求积分 $\int_0^1 \sin x dx$。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> q = IntSample('sin(x)',0,1,300)
q =   0.4624
```

理论值为 $\int_0^1 \sin x dx = 1-\cos 1 \approx 0.4597$，为了提高精度可以增大 n 的值：

```
>> q = IntSample('sin(x)',0,1,600)
q =   0.4610
```

8.9 平均抛物插值求积分

如果被积函数 $f(x)$ 是一个在下列离散点上给出值的函数：

$$x_0, x_1, \cdots, x_{n-1}, x_n$$
$$f_0, f_1, \cdots, f_{n-1}, f_n$$

且有 $a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b$，则

$$\int_a^b f(x)\mathrm{d}x = \frac{h_0}{6}(3f_0 + 3f_1 - R_0) + \frac{1}{6}\sum_{i=1}^{n-2} h_i(3f_i + 3f_{i+1} - \frac{L_i + R_i}{2})$$
$$+\frac{h_{n-1}}{6}(3f_{n-1} + 3f_n - L_{n-1})$$

其中上式中的符号的意义如下：

$$\begin{cases} L_i = \dfrac{\delta_i^2}{1+\delta_i} f_{i-1} - \delta_i f_i + \dfrac{\delta_i}{1+\delta_i} f_{i+1}, i = 1,2,\cdots,n-1 \\ R_i = \dfrac{\lambda_i}{1+\lambda_i} f_i - \lambda_i f_{i+1} + \dfrac{\lambda_i^2}{1+\lambda_i} f_{i+2}, i = 0,1,\cdots,n-2 \\ h_i = x_{i+1} - x_i, i = 0,1,\cdots,n-1 \\ \delta_i = h_i / h_{i-1}, i = 1,2,\cdots,n \\ \lambda_i = h_i / h_{i+1}, i = 0,1,\cdots,n-1 \end{cases}$$

在 MATLAB 中编程实现的抛物插值求积分的函数为：IntPWC

功能：用抛物插值求积分

调用格式：　q = IntPWC(X,Y,n)

其中，X：横坐标向量；

　　Y：纵坐标向量；

　　n：积分点的个数；

　　q：积分值。

抛物插值求积分的 MATLAB 代码如下所示：

```
function q = IntPWC(X,Y,n)
%横坐标向量：X
%纵坐标向量：Y
%积分点的个数：n
%积分结果：q
format long;
h = zeros(n-1,1);
lamda = zeros(n,1);
L = zeros(n-2,1);
R = zeros(n-2,1);
Dlta = zeros(n-2,1);
h(1) = X(2)-X(1);
for j=1:n-2            %计算每个子区间的积分参数
    h(j+1) = X(j+2)-X(j+1);
    lamda(j) = h(j)/h(j+1);
    Dlta(j) = h(j+1)/h(j);
    L(j) = (Dlta(j)*Dlta(j))*Y(j)/(1+Dlta(j))-Dlta(j)*Y(j+1)+…
          +Dlta(j)*Y(j+2)/(1+Dlta(j));
    R(j) = (lamda(j)*lamda(j))*Y(j+2)/(1+lamda(j))-lamda(j)*Y(j+1)+…
          +lamda(j)*Y(j)/(1+lamda(j));
end
```

```
q = h(1)*(3*Y(1)+3*Y(2)-R(1))/6+h(n-1)*(3*Y(n-1)+3*Y(n)-L(n-2))/6;
for k=2:n-2
    q = q + h(k)*(3*Y(k)+3*Y(k+1)-0.5*R(k)-0.5*L(k-1))/6;
end
format short;
```

例 8-17 平均抛物插值求积分应用实例。采用平均抛物插值求积分 $\int_0^1 \sin x\mathrm{d}x$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> x=0:0.1:1;
>> y=sin(x);
>> q=IntPWC(x,y,11)
q =  0.4597
```

从结果可以看出平均抛物插值的精度是很高的。

计算积分 $\int_0^1 \frac{\sin x}{x}\mathrm{d}x$，此积分是很经典的积分，用 MATLAB 的 Int 函数求得的结果为 0.94608307036718，用平均抛物插值求此积分的过程如下：

```
>> x=0:0.1:1
>> y(1)=1;
>> for j=2:11
     y(j)=sin(x(j))/x(j);
   end
>> q=IntPWC(x,y,11)
q =  0.9461
```

可见结果是很精确的。

8.10 奇异积分

对于一般的奇异积分，没有通用的数值方法来求出结果，只能针对具体的情况采取不同的方法。对奇异积分的处理方法有：积分变量替换法、奇异性的解析处理、分解法、康托洛维奇法、菲隆法等。

8.10.1 高斯-拉盖尔公式

高斯-拉盖尔公式有两种形式：

$$\int_0^{\infty} \mathrm{e}^{-x} f(x)\mathrm{d}x \approx \sum_{k=1}^{n} A_k f(x_k)$$

$$\int_0^{\infty} f(x)\mathrm{d}x \approx \sum_{k=1}^{n} A_k \mathrm{e}^{x_k} f(x_k)$$

系数和节点的值如表 8-5 所示。

表 8-5　高斯-拉盖尔公式节点和系数表

n	x_k	A_k
2	-0.585786	0.853553
	3.414214	0.146447
3	0.415575	0.711093
	2.294280	0.278518
	6.289945	0.0103893
4	0.322548	0.603154
	1.745761	0.357419
	4.536620	0.0388879
	9.395071	0.000539295
5	0.263560	0.521756
	1.413403	0.398667
	3.596426	0.0759424
	7.085810	0.00361176
	12.640801	0.0000233700

下面编制的程序是针对第一种形式的高斯-拉盖尔公式，即

$$\int_0^{\infty} e^{-x} f(x)dx \approx \sum_{k=1}^{n} A_k f(x_k)$$

因此程序的第一个输入参数——被积函数，是上式中的 $f(x)$，请注意例题的输入方式。

在 MATLAB 中编程实现的高斯-拉盖尔公式求积分的函数为：IntGaussLager

功能：用高斯-拉盖尔公式求积分

调用格式：q= IntGaussLager(f,n,AK,XK)

其中，f：被积函数；

n：所采用的积分点个数；

AK：自定义系数；

XK：自定义积分点；

q：积分值。

高斯-拉盖尔公式的 MATLAB 代码如下所示：

```
function q = IntGaussLager(f,n,AK,XK)
%被积函数：f;
%所采用的积分点个数：n;
%自定义系数：AK
%自定义积分点：XK;
%积分值：q;
if(n<6 && nargin == 2)
    AK = 0;
    XK = 0;
```

```
else
   q=sum(AK.*subs(sym(f),findsym(f),XK));
end
switch n
   case 2,                %n = 2 时，采用表 8-5 中对应的系数进行计算
      q=0.853553*subs(sym(f),findsym(sym(f)),-0.585786)+…
         0.146447*subs(sym(f),findsym(sym(f)),3.414214);
   case 3,                %n = 3 时，采用表 8-5 中对应的系数进行计算
      q=0.711093*subs(sym(f),findsym(sym(f)),0.415575)+…
         0.278518*subs(sym(f),findsym(sym(f)),2.294280)+…
         0.0103893*subs(sym(f),findsym(sym(f)),6.289945);
   case 4,                %n = 4 时，采用表 8-5 中对应的系数进行计算
      q=0.603154*subs(sym(f),findsym(sym(f)),0.322548)+…
         0.357419*subs(sym(f),findsym(sym(f)),1.745761)+…
         0.0388879*subs(sym(f),findsym(sym(f)),4.536620)+…
         0.000539295*subs(sym(f),findsym(sym(f)),9.395071);
   case 5,               %n = 5 时，采用表 8-5 中对应的系数进行计算
      q=0.521756*subs(sym(f),findsym(sym(f)),0.263560)+…
         0.398667*subs(sym(f),findsym(sym(f)),1.413403)+…
         0.0759424*subs(sym(f),findsym(sym(f)),3.596426)+…
         0.00361176*subs(sym(f),findsym(sym(f)),7.085810)+…
         0.0000233700*subs(sym(f),findsym(sym(f)),12.640801);
end
```

例 8-18　高斯-拉盖尔公式数值积分应用实例。计算积分 $\int_0^\infty e^{-x}\sin x dx$。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> q=IntGaussLager('sin(x)',4)
q =  0.5049
>> q=IntGaussLager('sin(x)',5)
q =  0.4989
```

所以由高斯-拉盖尔公式可得出 $\int_0^\infty e^{-x}\sin x dx \approx 0.4989$。

$\int_0^\infty e^{-x}\sin x dx$ 的准确结果为 0.5000，可见当 n 等于 5 时，高斯-拉盖尔公式的精度已经很高了。

8.10.2　高斯-埃尔米特公式

高斯-埃尔米特公式有以下两种形式：

$$\int_{-\infty}^{\infty} e^{-x^2} f(x)dx \approx \sum_{k=1}^{n} A_k f(x_k)$$

$$\int_{-\infty}^{\infty} f(x)dx \approx \sum_{k=1}^{n} A_k e^{x_k^2} f(x_k)$$

系数和节点的取值如表 8-6 所示。

表 8-6 高斯-埃尔米特公式节点和系数表

n	x_k	A_k
2	±0.707107	0.886227
3	0.000000	1.181636
	±1.224745	0.295409
4	±0.524648	0.544444
	±1.650680	0.100000
5	0.000000	0.945309
	±0.958572	0.393619
	±2.020183	0.199532

下面编制的程序是针对第一种形式的高斯-埃尔米特公式，即

$$\int_{-\infty}^{\infty} e^{-x^2} f(x)\mathrm{d}x \approx \sum_{k=1}^{n} A_k f(x_k)$$

因此程序的第一个输入参数——被积函数，是上式中的 $f(x)$，请注意例题的输入方式。

在 MATLAB 中编程实现的高斯-埃尔米特公式求积分的函数为：IntGaussHermite

功能：用高斯-埃尔米特公式求积分

调用格式：q= IntGaussHermite (f,n,AK,XK)

其中，f：被积函数；

n：所采用的积分点个数；

AK：自定义系数；

XK：自定义积分点；

q：积分值。

高斯-埃尔米特公式的 MATLAB 代码如下所示：

```
function q = IntGaussHermite(f,n,AK,XK)
%被积函数：f;
%所采用的积分点个数：n;
%自定义系数：AK
%自定义积分点：XK;
%积分值：q;
if(n<6 && nargin == 2)
   AK = 0;
   XK = 0;
else
   q=sum(AK.*subs(sym(f),findsym(sym(f)),XK));
end
switch n
   case 2,          %n = 2 时，采用表 8-6 中对应的系数进行计算
      q=0.886227*(subs(sym(f),findsym(sym(f)),-0.707107)+…
         subs(sym(f),findsym(f),0.707107));
   case 3,          %n = 3 时，采用表 8-6 中对应的系数进行计算
```

```
        q=1.181636*subs(sym(f),findsym(sym(f)),0)+…
            0.295409*(subs(sym(f),findsym(sym(f)),1.224745)+…
            subs(sym(f),findsym(sym(f)),-1.224745));
    case 4,          %n=4 时，采用表 8-6 中对应的系数进行计算
        q=0.544444*(subs(sym(f),findsym(sym(f)),0.524648)+…
            subs(sym(f),findsym(sym(f)),-0.524648))+...
            0.100000*(subs(sym(f),findsym(sym(f)),1.650680)+…
            subs(sym(f),findsym(sym(f)),-1.650680));
    case 5,          %n=5 时，采用表 8-6 中对应的系数进行计算
        q=0.945309*subs(sym(f),findsym(sym(f)),0)+…
            0.393619*(subs(sym(f),findsym(sym(f)),0.958572)+…
            subs(sym(f),findsym(sym(f)),-0.958572))+…
            0.199532*(subs(sym(f),findsym(sym(f)),2.020183)+…
            subs(sym(f),findsym(sym(f)),-2.020183));
end
```

例 8-19 高斯-埃尔米特公式数值积分应用实例。计算积分 $\int_{-\infty}^{\infty} e^{-x^2} dx$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> q=IntGaussHermite('1',4)
q =  1.2889
>> q=IntGaussHermite('1',5)
q =  2.1316
```

可见，由高斯-埃尔米特公式可得出 $\int_{-\infty}^{\infty} e^{-x^2} dx \approx 2.1316$。而积分 $\int_{-\infty}^{\infty} e^{-x^2} dx$ 的准确结果约为 1.7725，可见当 n 等于 5 时，高斯-埃尔米特公式的精度还比较差，要想提高精度，还得取更大的 n。

8.10.3 第一类切比雪夫积分

第一类切比雪夫积分的形式为：

$$\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} f(x)dx \approx \sum_{k=1}^{n} A_k f(x_k)$$

$$其中\ A_k = \frac{\pi}{n}, x_k = \cos\frac{2k-1}{2n}\pi$$

> 在 MATLAB 中编程实现的求第一类切比雪夫积分的函数为：IntQBXF1
> 功能：求第一类切比雪夫积分
> 调用格式：q = IntQBXF1(func,n)
> 其中，func：被积函数；
> n：积分所取项数；
> q：积分值。

第一类切比雪夫积分的 MATLAB 代码如下所示：

```
function q = IntQBXF1(func,n)
```

```
%被积函数：func
% 积分所取项数：n
%积分值：q
format long;
pi = 3.1415926535;
q = 0;
A = zeros(n,1);
x = zeros(n,1);
for i=1:n
    A(i) = pi/n;
    x(i) = cos(pi*(2*i-1)/2/n);
    y(i) = subs(sym(func), findsym(sym(func)), x(i));
    q = q + A(i)*y(i);
end
```

例 8-20 第一类切比雪夫积分应用实例。采用第一类切比雪夫积分，计算积分 $\int_{-1}^{1}\frac{1}{\sqrt{1-x^2}}\mathrm{d}x$。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> q=IntQBXF1('1',10)
q =   3.14159265350000
```

而 $\int_{-1}^{1}\frac{1}{\sqrt{1-x^2}}\mathrm{d}x=\pi$，可与计算结果相比较。

计算积分 $\int_{-1}^{1}\frac{1}{\sqrt{1-x^2}}\mathrm{d}x$，其理论值为 $\frac{\pi}{2}$，用第一类切比雪夫积分求得：

```
>> q=IntQBXF1('x^2',50)
q =  1.57079632670507
```

结果正好是 $\frac{\pi}{2}$。

8.10.4 第二类切比雪夫积分

第二类切比雪夫积分的形式为：

$$\int_{-1}^{1}\sqrt{1-x^2}f(x)\mathrm{d}x\approx\sum_{k=1}^{n}A_k f(x_k)$$

$$\text{其中 } A_k=\frac{\pi}{n+1}\sin^2\left(\frac{k\pi}{n+1}\right), x_k=\cos\frac{k\pi}{n+1}$$

在 MATLAB 中编程实现的求第二类切比雪夫积分的函数为：IntQBXF2

功能：求第二类切比雪夫积分

调用格式：q = IntQBXF2(func,n)

其中，func：被积函数；

n：积分所取项数；

q：积分值。

第二类切比雪夫积分的 MATLAB 代码如下所示：

```
function q = IntQBXF2(func,n)
%被积函数：func
% 积分所取项数：n
%积分值：q
format long;
pi = 3.1415926535;
q = 0;
A = zeros(n,1);
x = zeros(n,1);
for i=1:n
    A(i) = sin((i*pi)/(n+1))*sin((i*pi)/(n+1))*pi/(n+1);
    x(i) = cos(pi*i/(n+1));
    y(i) = subs(sym(func), findsym(sym(func)), x(i));
    q = q + A(i)*y(i);
end
```

例 8-21 第二类切比雪夫积分应用实例。采用第二类切比雪夫积分，计算积分 $\int_{-1}^{1}\sqrt{1-x^2}\mathrm{d}x$。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> q=IntQBXF2('1',10)
q =   1.57079632679367
```

而 $\int_{-1}^{1}\sqrt{1-x^2}\mathrm{d}x=\dfrac{\pi}{2}$，可与计算结果相比较。

计算积分 $\int_{-1}^{1}x^2\sqrt{1-x^2}\mathrm{d}x$，其理论值为 $\dfrac{\pi}{8}$，用第二类切比雪夫积分求得：

```
>> q=IntQBXF2('x^2',50)
q =   0.39269908169748
```

结果的精度达到 11 位数字。

8.11 重积分的数值计算

对积分区域为矩形的重积分的数值计算与定积分相似，相应的有梯形公式和辛普森公式等。以下的重积分的数值计算都基于矩形的积分区域 $a\leqslant x\leqslant A,b\leqslant y\leqslant B$。

8.11.1 梯形公式

对二重积分 $\int_b^B\int_a^A f(x,y)\mathrm{d}x\mathrm{d}y$ 有如下的梯形公式来求解：

$$\int_b^B\int_a^A f(x,y)\mathrm{d}x\mathrm{d}y \approx \frac{(B-b)(A-a)}{4}[f(a,b)+f(A,b)+f(a,B)+f(A,B)]$$

梯形公式的精度显然不高，要想提高精度，可采用下面的复合梯形公式：

$$\int_b^B\int_a^A f(x,y)\mathrm{d}x\mathrm{d}y \approx \frac{(B-b)(A-a)}{4mn}\sum_{i=0}^{n}\sum_{j=0}^{m}C_{ij}f(x_i,y_j)]$$

其中 $x_i = a+\frac{i(A-a)}{n}, y_j = b+\frac{j(B-b)}{m}$

$$[C]=\begin{bmatrix} 1 & 2 & 2 & \cdots & 2 & 2 & 1 \\ 2 & 4 & 4 & \cdots & 4 & 4 & 2 \\ 2 & 4 & 4 & \cdots & 4 & 4 & 2 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 2 & 4 & 4 & \cdots & 4 & 4 & 2 \\ 2 & 4 & 4 & \cdots & 4 & 4 & 2 \\ 1 & 2 & 2 & \cdots & 2 & 2 & 1 \end{bmatrix}_{(n+1)(m+1)}$$

在 MATLAB 中编程实现的梯形公式求重积分的函数为：DblTraprl

功能：用梯形公式求重积分

调用格式：q=DblTraprl(f,a,A,b,B,m,n)

其中，func：被积函数；

a：X 积分区间左端点；

A：X 积分区间右端点；

b：Y 积分区间左端点；

B：Y 积分区间右端点；

m：X 方向子区间数目；

n：Y 方向子区间数目；

q：积分值。

重积分梯形公式的 MATLAB 代码如下所示：

```
function q=DblTraprl(f,a,A,b,B,m,n)
%被积函数：f
% X积分区间左端点：a
% X积分区间右端点：A
% Y积分区间左端点：b
% Y积分区间右端点：B
% X方向子区间数目：m
% Y方向子区间数目：n
%积分值：q
if(m==1 && n==1)              %梯形公式
    q=((B-b)*(A-a)/4)*(subs(sym(f),findsym(sym(f)),{a,b})+…
       subs(sym(f),findsym(sym(f)),{a,B})+…
       subs(sym(f),findsym(sym(f)),{A,b})+…
```

```
        subs(sym(f),findsym(sym(f)),{A,B}));
else                        %复合梯形公式
    C=4*ones(n+1,m+1);
    C(1,:)=2;
    C(:,1)=2;
    C(n+1,:)=2;
    C(:,m+1)=2;
    C(1,1)=1;
    C(1,m+1)=1;
    C(n+1,1)=1;
    C(n+1,m+1)=1;           %C 矩阵
end
F=zeros(n+1,m+1);
q=0;
for i=0:n
   for j=0:m
       x=a+i*(A-a)/n;
       y=b+j*(B-b)/m;
       F(i+1,j+1)=subs(sym(f),findsym(sym(f)),{x,y});
       q=q+F(i+1,j+1)*C(i+1,j+1);
   end
end
q=((B-b)*(A-a)/4/m/n)*q;
```

例 8-22 复合梯形公式计算重积分应用实例。用复合梯形公式计算重积分 $\int_0^1\int_0^1 \sin(xy)\mathrm{d}x\mathrm{d}y$。

解：在 MATLAB 命令窗口中输入下列命令：

```
>> q=DblTraprl('sin(x*y)',0,1,0,1,15,10)
q =  0.2397
```

所以 $\int_0^1\int_0^1 \sin(xy)\mathrm{d}x\mathrm{d}y \approx 0.2397$。

8.11.2 辛普森公式

对二重积分 $\int_b^B\int_a^A f(x,y)dxdy$ 有如下的辛普森公式来求解：

$$\int_b^B\int_a^A f(x,y)dxdy \approx \frac{(B-b)(A-a)}{9}[f(a,b)+f(a,B)+f(A,b)$$
$$+f(A,B)+4f(\frac{A-a}{2},b)+4f(a,\frac{B-b}{2})+4f(\frac{A-a}{2},B)$$
$$+4f(A,\frac{B-b}{2})+16f(\frac{A-a}{2},\frac{B-b}{2})]$$

同样有精度更高的复合辛普森公式：

$$\int_b^B\int_a^A f(x,y)\mathrm{d}x\mathrm{d}y \approx \frac{(B-b)(A-a)}{36mn}\sum_{i=0}^{2n}\sum_{j=0}^{2m} C_{ij} f(x_i,y_j)]$$

其中 n,m 分别代表 x 方向子区间数目的一半和 y 方向子区间数目的一半且有 $x_i = a + \dfrac{i(A-a)}{2n}, y_j = b + \dfrac{j(B-b)}{2m}$

$$[C] = \begin{bmatrix} 1 & 4 & 2 & 4 & 2 & \cdots & 4 & 2 & 4 & 1 \\ 4 & 16 & 8 & 16 & 8 & \cdots & 16 & 8 & 16 & 4 \\ 2 & 8 & 4 & 8 & 4 & \cdots & 8 & 4 & 8 & 2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 2 & 8 & 4 & 8 & 4 & \cdots & 8 & 4 & 8 & 2 \\ 4 & 16 & 8 & 16 & 8 & \cdots & 16 & 8 & 16 & 8 \\ 1 & 4 & 2 & 4 & 2 & \cdots & 4 & 2 & 4 & 1 \end{bmatrix}_{(2n+1)(2m+1)}$$

在 MATLAB 中编程实现的辛普森公式求重积分的函数为：DblSimpson

功能：用辛普森公式求重积分

调用格式：q=DblSimpson(f,a,A,b,B,m,n)

其中，func：被积函数；

a：X 积分区间左端点；

A：X 积分区间右端点；

b：Y 积分区间左端点；

B：Y 积分区间右端点；

m：X 方向子区间数目的一半；

n：Y 方向子区间数目的一半；

q：积分值。

重积分辛普森公式的 MATLAB 代码如下所示：

```
function q=DblSimpson(f,a,A,b,B,m,n)
%被积函数：func
% X积分区间左端点：a
% X积分区间右端点：A
% Y积分区间左端点：b
% Y积分区间右端点：B
% X方向子区间数目的一半：m
% Y方向子区间数目的一半：n
%积分值：q
if(m==1 && n==1)              %辛普森公式
    q=((B-b)*(A-a)/9)*(subs(sym(f),findsym(sym(f)),{a,b})+…
        subs(sym(f),findsym(sym(f)),{a,B})+…
        subs(sym(f),findsym(sym(f)),{A,b})+…
        subs(sym(f),findsym(sym(f)),{A,B})+…
        4*subs(sym(f),findsym(sym(f)),{(A-a)/2,b})+…
        4*subs(sym(f),findsym(sym(f)),{(A-a)/2,B})+…
        4*subs(sym(f),findsym(sym(f)),{a,(B-b)/2})+…
        4*subs(sym(f),findsym(sym(f)),{A,(B-b)/2})+…
```

```
        16*subs(sym(f),findsym(sym(f)),{(A-a)/2,(B-b)/2}));
    else                        %复合辛普森公式
        q=0;
        for i=0:n-1
            for j=0:m-1
                x=a+2*i*(A-a)/2/n;
                y=b+2*j*(B-b)/2/m;
                x1=a+(2*i+1)*(A-a)/2/n;
                y1=b+(2*j+1)*(B-b)/2/m;
                x2=a+2*(i+1)*(A-a)/2/n;
                y2=b+2*(j+1)*(B-b)/2/m;
                q=q+subs(sym(f),findsym(sym(f)),{x,y})+…
                    subs(sym(f),findsym(sym(f)),{x,y2})+…
                    subs(sym(f),findsym(sym(f)),{x2,y})+…
                    subs(sym(f),findsym(sym(f)),{x2,y2})+…
                    4*subs(sym(f),findsym(sym(f)),{x,y1})+…
                    4*subs(sym(f),findsym(sym(f)),{x2,y1})+…
                    4*subs(sym(f),findsym(sym(f)),{x1,y})+…
                    4*subs(sym(f),findsym(sym(f)),{x1,y2})+…
                    16*subs(sym(f),findsym(sym(f)),{x1,y1});
            end
        end
    end
    q=((B-b)*(A-a)/36/m/n)*q;
```

例 8-23 复合辛普森公式计算重积分应用实例。用复合辛普森公式计算重积分 $\int_0^1\int_0^1 e^x \sin(xy)\mathrm{d}x\mathrm{d}y$。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> q=DblSimpson('exp(x)*sin(x*y)',0,1,0,1,15,15)
q =  0.4771
```

所以 $\int_0^1\int_0^1 \mathrm{e}^x \sin(xy)dxdy \approx 0.4771$。

8.11.3 高斯公式

$$\int_{-1}^1\int_{-1}^1 f(x,y)\mathrm{d}x\mathrm{d}y \approx \sum_{i=1}^m\sum_{j=1}^n A_iA_jf(x_i,y_j)$$

$$=[A_1 \quad A_2 \quad \cdots \quad A_n]\begin{bmatrix} f(x_1,y_1) & f(x_2,y_1) & \cdots & f(x_m,y_1) \\ f(x_1,y_2) & f(x_2,y_2) & \cdots & f(x_m,y_2) \\ \vdots & \vdots & \ddots & \vdots \\ f(x_1,y_n) & f(x_2,y_n) & \cdots & f(x_m,y_n) \end{bmatrix}\begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_m \end{bmatrix}$$

在 MATLAB 中编程实现的高斯公式求重积分的函数为：IntDBGauss

功能：用高斯公式求重积分

调用格式：q = IntDBGauss(func,a,b,c,d,m,n)

其中，func：被积函数；
a：X 积分区间左端点；
b：X 积分区间右端点；
c：Y 积分区间左端点；
d：Y 积分区间右端点；
m：X 方向所采用的高斯积分点个数；
n：Y 方向所采用的高斯积分点个数；
q：积分值。

重积分高斯公式的 MATLAB 代码如下所示：

```
function q = IntDBGauss(func,a,b,c,d,m,n)
%被积函数：func
% X积分区间左端点：a
% X积分区间右端点：b
% Y积分区间左端点：c
% Y积分区间右端点：d
% X方向所采用的高斯积分点个数：m
% Y方向所采用的高斯积分点个数：n
%积分值：q
format long;
ta = (b-a)/2;
tb = (a+b)/2;
tc = (d-c)/2;
td = (c+d)/2;
f = zeros(n,m);
node_x = [0 0 0 0 0;-0.5773503  0.5773503 0 0 0;-0.7745967  0  0.7745967  0 0;
        -0.8611363  -0.3398810  0.3398810  0.8611363 0;
        -0.9061793  -0.5384693  0  0.5384693  0.9061793];  %点坐标矩阵
node_A = [2  0  0  0  0;1  1  0  0  0;0.5555556  0.8888889  0.5555556   0  0;
         0.3478548  0.6521453   0.6521453  0.3478548  0;
         0.2369269  0.4786287  0.5688889  0.4786287  0.2369269];%系数矩阵
for i=1:m
   for j=1:n
      f(j,i) = subs(sym(func),findsym(sym(func)),[ta*node_x(m,i)+tb
tc*node_x(n,i)+td]);
   end
end
Am = transpose(node_A(m,1:m));
An = node_A(n,1:n);
q =ta*tc* An*f*Am;
format short;
```

例 8-24 高斯公式求重积分应用实例。用高斯公式计算重积分 $\int_0^1\int_0^1 e^{-x}\sin(xy)\mathrm{d}x\mathrm{d}y$ 。

解： 在 MATLAB 命令窗口中输入下列命令：

```
>> q=IntDBGauss('exp(-x)*sin(x*y)',0,1,0,1,2,2)
q =     0.1505
```

所以用两个积分点算得 $\int_0^1\int_0^1 e^{-x}\sin(xy)\mathrm{d}x\mathrm{d}y \approx 0.1505$。

8.12 小结

数值积分相对数值微分要容易些，因此求数值积分的方法比数值微分要多，而且有很多精度十分高的数值积分方法还正在不断发展中。

第 9 章　方程求根

方程求根作为数学一个古老的研究领域已有几百年了，不少数学家都涉猎过这个领域，包括牛顿、贝努利等，因此关于方程求根的方法非常多，特别是多项式的求根研究得非常透彻。在求方程的根之前，首先要判断方程根的存在性和根的个数，但是在实际应用中我们更关心的是在某个感兴趣的区间上是否存在根的问题，因此，本章的算法大都是在已知区间上求根。

通过本章的学习，读者不仅能掌握常见的方程求根算法，而且还能熟练使用 MATLAB 编程来实现这些算法。

9.1 方程的基本理论

数学问题中经常遇见的方程为代数多项式方程和超越方程。代数方程的基本形式为：

$$f(x) = x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n = 0$$

上式中的 $a_1, a_2, \cdots, a_n$ 为实常数，而超越方程的形式则多种多样。

求方程的根的基本思想是把方程的根分割在某些区间内，然后在这些区间内求出方程的一个根。其理论依据为：如果 $f(a)f(b) < 0$ ，则在区间 $[a,b]$ 上方程 $f(x) = 0$ 至少存在一个实根。

9.2 方程求根的数值方法

在根存在的前提下，要求方程的根，先要确定根的界限，即根的上下界。然后把根隔离成一个一个的区间，在每个区间上再迭代逼近根，也就是缩小根存在的区间范围。因此先介绍确定代数多项式方程根的上下界的贝努利法，然后介绍求方程根的其他数值方法。

9.2.1 贝努利法

贝努利法是用来求如下方程：

$$f(x) = x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n = 0(a_n \neq 0)$$

按模最大最小实根。

9.2.1.1 按模最大实根

采用贝努利法求按模最大实根的算法步骤如下：

❶ 用方程的系数取反号按升幂次序排列，做一个计算 $f(k)$ 的模板：

$$-a_n, -a_{n-1}, \cdots, -a_2, -a_1$$

为计算

$$f(1), f(2), \cdots, f(n-1), f(n), \cdots$$

取

$$f(1)=0, f(2)=0, \cdots, f(n-1)=0, f(n)=1$$

对 $k=1,2,\cdots$

❷ 令

$$f(n+1) = -a_n \bullet f(1) - a_{n-1} \bullet f(2) - \cdots - a_1 \bullet f(n)$$

然后再令

$$f(1)=f(2), f(2)=f(3), \cdots, f(n)=f(n+1)$$

$$x_k = \frac{f(n)}{f(n-1)}$$

如此当 $|x_k - x_{k-1}| < \varepsilon$，则 x_k 为按模最大的实根。

> 在 MATLAB 中编程实现的贝努利法求按模最大实根的函数为：BenvliMAX
>
> 功能：贝努利法求按模最大实根
>
> 调用格式：xm = BenvliMAX(a,eps)
>
> 其中，a：方程系数；
>
> eps：根的精度；
>
> xm：求得的按模最大实根。

贝努利法求按模最大实根的 MATLAB 程序代码如下所示：

```
function xm = BenvliMAX(a,eps)
%方程系数：a
%根的精度：eps
%求得的按模最大实根：xm
format long;
if(nargin==1)
   eps=1.0e-4;
end
n = length(a);
y = zeros(n+1,1);
y(n) =1;
tol = 1;
xm = inf;
while tol>eps
    xm1 = xm;
    y(n+1) =dot(-a,y(1:n));
    y(1:n) = y(2:n+1);
    xm = y(n)/y(n-1);
    tol = abs(xm-xm1);
end
format short;
```

例 9-1　贝努利法求按模最大实根应用实例。求方程 $x^3+3.5x^2-5x-12=0$ 的按模最大实根。

解： 在 MATLAB 命令窗口中输入：

```
>> a=[-12 -5 3.5];              %注意系数的顺序——升序排列
>> xm = BenvliMAX(a)
xm =   -4.0000
```

可见，方程 $x^3+3.5x^2-5x-12=0$ 的按模最大实根为 $x=-4$。

实际上，$x^3+3.5x^2-5x-12=(x-2)(x+4)(x+3.5)=0$，所以可以直接看出结果。

9.2.1.2　按模最小实根

采用贝努利法求按模最小实根的算法步骤如下：

❶ 用方程的系数取反号按升幂次序排列，做一个计算 $f(k)$ 的模板：

$$-\frac{1}{a_n},-\frac{a_1}{a_n},\cdots,-\frac{a_{n-2}}{a_n},-\frac{a_{n-1}}{a_n}$$

为计算

$$f(1),f(2),\cdots,f(n-1),f(n),\cdots$$

取

$$f(1)=0,f(2)=0,\cdots,f(n-1)=0,f(n)=1$$

对 $k=1,2,\cdots$

❷ 令

$$f(n+1)=-a_nf(1)-a_{n-1}f(2)-\cdots-a_1f(n)$$

然后再令

$$f(1)=f(2),f(2)=f(3),\cdots,f(n)=f(n+1)$$

$$x_k=\frac{f(n-1)}{f(n)}$$

如此当 $|x_k-x_{k-1}|<\varepsilon$，则 x_k 为按模最小的实根。

> 在 MATLAB 中编程实现的贝努利法求按模最小实根的函数为：BenvliMIN
>
> 功能：贝努利法求按模最小实根
>
> 调用格式：xm = BenvliMIN(a,eps)
>
> 其中，a：方程系数；
>
> 　　　eps：根的精度；
>
> 　　　xm：求得的按模最小实根。

贝努利法求按模最小实根的 MATLAB 程序代码如下所示：

```
function xm = BenvliMIN(a,eps)
%方程系数：a
%根的精度：eps
```

```
%求得的按模最大实根: xm
format long;
if(nargin==1)
    eps=1.0e-4;
end
n = length(a);
y = zeros(n+1,1);
y(n) =1;
am(1:n) = a(n:-1:1);
c = am(n);
am(2:n) = am(1:n-1);
am(1) = 1;
am(1:n) = -am(1:n)/c;
tol = 1;
xm = inf;
while tol>eps
    xm1 = xm;
    y(n+1) =dot(am,y(1:n));
    y(1:n) = y(2:n+1);
    xm =y(n-1)/y(n);
    tol = abs(xm-xm1);
end
format short;
```

例 9-2 贝努利法求按模最小实根应用实例。求方程 $x^3+3.5x^2-5x-12=0$ 的按模最小实根。

解: 在 MATLAB 命令窗口中输入:

```
>> a=[-12 -5 3.5];              %注意系数的顺序——升序排列
>> xm = BenvliMIN(a)
xm =   -1.5000
```

可见，方程 $x^3+3.5x^2-5x-12=0$ 的按模最小实根为 $x=-1.5$。

实际上，$x^3+3.5x^2-5x-12=(x-2)(x+4)(x+1.5)=0$，所以也可以直接看出结果。

9.2.2 二分法

二分法的具体求解步骤介绍如下。

❶ 计算函数 $f(x)$ 在区间 $[a,b]$ 中点的函数值 $f(\frac{a+b}{2})$，并作下面的判断:

如果 $f(a)f(\frac{a+b}{2})<0$，转到❷;

如果 $f(a)f(\frac{a+b}{2})>0$，令 $a=\frac{a+b}{2}$，转到❶;

如果 $f(a)f(\frac{a+b}{2})=0$，则 $x=\frac{a+b}{2}$ 为一个根。

❷ 如果 $\left|a-\frac{a+b}{2}\right|<\varepsilon$（预先给定的精度），则 $x=\frac{b+3a}{4}$ 为一个根，否则令 $b=\frac{a+b}{2}$，

转到❶。

在 MATLAB 中编程实现的二分法的函数为：HalfInterval
功能：用二分法求方程的一个根
调用格式：root=HalfInterval(f,a,b,eps)
其中，f：方程名；
a：区间左端点；
b：区间右端点；
eps：根的精度；
root：求得的根。

二分法的 MATLAB 程序代码如下所示：

```
function root=HalfInterval(f,a,b,eps)
%方程表达式：f
%区间左端点：a
%区间右端点：b
%根的精度：eps
%求得的根：root
if(nargin==3)
    eps=1.0e-4;
end
f1=subs(sym(f),findsym(sym(f)),a);                 %两端点的函数值
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
    root=a;
end
if(f2==0)
    root=b;
end
if(f1*f2>0)
    disp('两端点函数值乘积大于0!');
    return;
else
    root=FindRoots(f,a,b,eps);                     %调用求解子程序
end
function r=FindRoots(f,a,b,eps)
f_1=subs(sym(f),findsym(sym(f)),a);
f_2=subs(sym(f),findsym(sym(f)),b);
mf=subs(sym(f),findsym(sym(f)),(a+b)/2);           %中点函数值
if(f_1*mf>0)
    t=(a+b)/2;
    r=FindRoots(f,t,b,eps);                        %右递归
else
    if(f_1*mf==0)
        r=(a+b)/2;
    else
        if(abs(b-a)<=eps)
```

```
        r=(b+3*a)/4;                          %输出根
      else
        s=(a+b)/2;
        r=FindRoots(f,a,s,eps);               %左递归
      end
    end
end
```

例 9-3 二分法求根应用实例。采用二分法求方程 $x^3-3x+1=0$ 在区间[0,1]上的一个根。

解： 在 MATLAB 命令窗口中输入：

```
>> r=HalfInterval('x^3-3*x+1',0,1)
```

输出计算结果为：

```
r =    0.3473
```

可见，方程 $x^3-3x+1=0$ 在区间[0,1]上的一个根为 $x=0.3473$。

9.2.3 黄金分割法

二分法是把求解区间的长度逐次减半，而黄金分割法是把求解区间逐次缩短为前次的 0.618 倍。它的求解步骤介绍如下。

❶ 设 $t_1=a+(1-0.618)\times(b-a)$，$t_2=a+0.618\times(b-a)$，且 $f_1=f(t_1)$，$f_2=f(t_2)$。

❷ 如果 $|t_1-t_2|<\varepsilon$（给定的最小区间长度），则输出方程的根 $\frac{t_1+t_2}{2}$；否则转到❸。

❸ 如果 $f_1\times f_2<0$，则令 $a=t_1$，$b=t_2$，转❶，否则如果 $f_1\times f(a)>0$，令 $a=t_2$，反之令 $b=t_1$，转到❶。

在 MATLAB 中编程实现的黄金分割法的函数为：hj

功能：用黄金分割法求方程的一个根

调用格式：root= hj (f,a,b,eps)

其中，f：方程名；

a：区间左端点；

b：区间右端点；

eps：根的精度；

root：求得的根。

黄金分割法的 MATLAB 程序代码如下所示：

```
function root=hj(f,a,b,eps)
%方程表达式：f
%区间左端点：a
%区间右端点：b
%根的精度：eps
%求得的根：root
```

```
if(nargin==3)
   eps=1.0e-4;
end
f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
   root=a;
end
if(f2==0)
   root=b;
end
if(f1*f2>0)
   disp('两端点函数值乘积大于 0!');
   return;
else
   t1=a+(b-a)*0.382;
   t2=a+(b-a)*0.618;
   f_1=subs(sym(f),findsym(sym(f)),t1);
   f_2=subs(sym(f),findsym(sym(f)),t2);
   tol=abs(t1-t2);
   while(tol>eps)                          %精度控制
      if(f_1*f_2<0)
         a=t1;                             %区间两端点都调整
         b=t2;
      else
         fa=subs(sym(f),findsym(sym(f)),a);
         if(f_1*fa>0)
            a=t2;                          %同号左端点调整
         else
            b=t1;                          %异号右端点调整
         end
      end
      t1=a+(b-a)*0.382;
      t2=a+(b-a)*0.618;
      f_1=subs(sym(f),findsym(sym(f)),t1);
      f_2=subs(sym(f),findsym(sym(f)),t2);
      tol=abs(t2-t1);
   end
   root=(t1+t2)/2;                         %输出根
end
```

例 9-4 黄金分割法求根应用实例。采用黄金分割法求方程 $x^3-3x+1=0$ 在区间[0,1]上的一个根。

解：在 MATLAB 命令窗口中输入：

```
>> r=hj('x^3-3*x+1',0,1)
```

输出计算结果为：

```
r =    0.3473
```

可见，方程 $x^3-3x+1=0$ 在区间[0,1]上的一个根为 $x=0.3473$，和二分法求出来的结果相同。

9.2.4 不动点迭代法

要求方程 $f(x)=0$ 的根，可以先把方程改写成如下的形式：

$$x=f(x)+x$$

于是得到不动点迭代法的其中一种迭代公式：

$$x_n=f(x_{n-1})+x_{n-1}$$

此种不动点迭代法很有可能不收敛，因为它的本质是求函数 $y=f(x)+x$ 与直线 $y=x$ 的交点，而它们不一定存在交点。即使收敛，其速度也十分慢，因此有了艾肯特加速迭代与史蒂芬森加速迭代。

在 MATLAB 中编程实现的不动点迭代法的函数为：StablePoint

功能：用不动点迭代法求方程的一个根

调用格式：[root,n]=StablePoint(f,x0,eps)

其中，f：方程名；

x0：初始迭代值；

eps：根的精度；

root：求得的根；

n：迭代步数。

不动点迭代法的 MATLAB 程序代码如下所示：

```
function [root,n]=StablePoint(f,x0,eps)
%方程表达式：f
%初始迭代值：x0
%根的精度：eps
%求得的根：root
%迭代步数：n
if(nargin==2)
   eps=1.0e-4;
end
tol=1;
root=x0;
n=0;
while(tol>eps)
    n=n+1;
    r1=root;
    root=subs(sym(f),findsym(sym(f)),r1)+r1;            %迭代的核心公式
    tol=abs(root-r1);
end
```

例 9-5 不动点迭代法求根应用实例。用不动点迭代法求方程 $\frac{1}{\sqrt{x}}+x-2=0$ 的一个根，迭代初始值为 0.5。

解： 在 MATLAB 命令窗口中输入：

```
>> [r,n]=StablePoint('1/sqrt(x)+x-2',0.5)
r =    0.3820
n =    4
```

从计算结果可以看出，经过 4 步迭代，得出方程 $\frac{1}{\sqrt{x}}+x-2=0$ 的一个根为 $x=0.3820$。

9.2.4.1　艾肯特加速

艾肯特加速是在计算出 x_n、x_{n+1}、x_{n+2} 后，对 x_{n+1} 作以下修正：

$$\hat{x}_{n+1} = x_n - \frac{(x_{n+1} - x_n)^2}{x_{n+2} - 2x_{n+1} + x_n}$$

然后用 $\hat{x}_{n+1}$ 来逼近方程的根。

艾肯特加速是先用不动点迭代法算出一系列 $\{x_n\}$，再对此系列作修正得到系列 $\left\{\hat{x}_n\right\}$，用后者来逼近方程的根。

> 在 MATLAB 中编程实现的艾肯特加速的函数为：AtkenStablePoint
> 功能：用艾肯特加速的不动点迭代法求方程的一个根
> 调用格式：[root,n]= AtkenStablePoint (f,x0,eps)
> 其中，f：方程名；
> 　　x0：初始迭代值；
> 　　eps：根的精度；
> 　　root：求得的根；
> 　　n：迭代步数。

艾肯特加速不动点迭代法的 MATLAB 程序代码如下所示：

```
function [root,n]=AtkenStablePoint(f,x0,eps)
%方程表达式：f
%初始迭代值：x0
%根的精度：eps
%求得的根：root
%迭代步数：n
if(nargin==2)
   eps=1.0e-4;
end
tol=1;
root=x0;
x(1:2)=0;
n=0;
m=0;
a2=x0;
while(tol>eps)
   n=n+1;
   a1=a2;
```

```
    r1=root;
    root=subs(sym(f),findsym(sym(f)),r1)+r1;          %求出根
    x(n)=root;                                        %把所有的根都保存下来
    if(n>2)
        m=m+1;
        a2=x(m)-(x(m+1)-x(m))^2/(x(m+2)-2*x(m+1)+x(m));     %对保存的根进行艾肯特
%修正
        tol=abs(a2-a1);
    end
end
root=a2;
```

例 9-6 艾肯特加速不动点迭代法求根应用实例。用艾肯特加速迭代法求方程 $\frac{1}{\sqrt{x}}+x-2=0$ 的一个根，迭代初始值为 0.5。

解： 在 MATLAB 命令窗口中输入：

```
>> [r,n]= AtkenStablePoint('1/sqrt(x)+x-2',0.5)
r =    0.3820
n =    4
```

从计算结果可以看出，经过 4 步迭代，得出方程 $\frac{1}{\sqrt{x}}+x-2=0$ 的一个根为 $x=0.3820$。这个方程比较简单，所以通过简单的几步迭代就可得出根，下面的结果可以看出艾肯特加速迭代法的优点。

```
>> [r,n]=AtkenStablePoint('1/sqrt(x)+x-2',0.999) %，迭代初始值为 0.999
r =    1.0000
n =    4
>> [r,n]=StablePoint('1/sqrt(x)+x-2',0.999)  %，迭代初始值为 0.999
r =    0.3820
n =    21
```

上面的例子说明采用初始值 0.999 时，经过 4 步迭代艾肯特加速迭代法，可以得到方程 $\frac{1}{\sqrt{x}}+x-2=0$ 的另一个根 $x=1$，而普通的不动点迭代法却得不到。

9.2.4.2 史蒂芬森加速

史蒂芬森加速与艾肯特加速不同的地方在于前者是在迭代的同时就进行修正，它的迭代公式为：

$$y_n = f(x_n)+x_n$$
$$z_n = f(y_n)+y_n$$
$$x_{n+1} = x_n - \frac{(y_n-x_n)^2}{z_n-2y_n+x_n}$$

其中迭代初始值为 x_0。

> 在 MATLAB 中编程实现的史蒂芬森加速的函数为：StevenStablePoint

功能：用史蒂芬森加速的不动点迭代法求方程的一个根
调用格式：[root,n]= StevenStablePoint (f,x0,eps)
其中，f：方程名；
x0：初始迭代值；
eps：根的精度；
root：求得的根；
n：迭代步数。

史蒂芬森加速不动点迭代法的 MATLAB 程序代码如下所示：

```
function [root,n]=StevenStablePoint(f,x0,eps)
%方程表达式：f
%初始迭代值：x0
%根的精度：eps
%求得的根：root
%迭代步数：n
if(nargin==2)
    eps=1.0e-4;
end
tol=1;
root=x0;
n=0;
while(tol>eps)
    n=n+1;
    r1=root;
    y=subs(sym(f),findsym(sym(f)),r1)+r1;
    z=subs(sym(f),findsym(sym(f)),y)+y;
    root=r1-(y-r1)^2/(z-2*y+r1);            %对每次算出的根立即进行修正
    tol=abs(root-r1);
end
```

例 9-7 史蒂芬森加速不动点迭代法求根应用实例。用史蒂芬森加速迭代法求方程 $\frac{1}{\sqrt{x}}+x-2=0$ 的一个根。

解：在 MATLAB 命令窗口中输入：

```
>> [r,n]=StevenStablePoint('1/sqrt(x)+x-2',0.999) %，迭代初始值为 0.999
r =    1.0000
n =    2
>> [r,n]=StevenStablePoint('1/sqrt(x)+x-2',0.5) %，迭代初始值为 0.5
r =    0.3820
n =    4
```

从例子可以看出，史蒂芬森加速迭代法不仅能求出方程 $\frac{1}{\sqrt{x}}+x-2=0$ 的一个根 $x=0.3820$，而且它比艾肯特加速迭代法更快地求出另一个根 $x=1$。

9.2.5 弦截法

弦截法是一种不必进行导数运算的求根方法。弦截法在迭代过程中不仅用到前一步 x_{k-1} 处的函数值，而且还使用 x_k 处的函数值来构造迭代函数，这样做能提高迭代的收敛速度。

下面讲述 5 种常见的弦截法，它们分别是：一般弦截法、单点弦截法、双点弦截法、平行弦截法和改进弦截法。

9.2.5.1 一般弦截法

一般弦截法的算法过程如下：

❶ 过两点 $(a,f(a))$ 与 $(b,f(b))$ 作一直线，它与 X 轴有一个交点，记为 x_1；

❷ 如果 $f(a)f(x_1)<0$，过两点 $(a,f(a))$ 与 $(x_1,f(x_1))$ 作一直线，它与 X 轴的交点记为 x_2，否则过两点 $(b,f(b))$，$(x_1,f(x_1))$ 作一直线，它与 X 轴的交点记为 x_2；

❸ 如此下去，直到 $|x_n-x_{n-1}|<\varepsilon$，就可以认为 x_n 为 $f(x)=0$ 在区间 $[a,b]$ 的一个根；

❹ x_k 的递推公式为：

$$\begin{cases} x_k=a-\dfrac{x_{k-1}-a}{f(x_{k-1})-f(a)}f(a) & f(a)f(x_{k-1})<0 \\ x_k=b-\dfrac{x_{k-1}-b}{f(x_{k-1})-f(b)}f(b) & f(a)f(x_{k-1})>0 \end{cases}$$

且 $x_1=a-\dfrac{b-a}{f(b)-f(a)}f(a)$。

> 在 MATLAB 中编程实现的一般弦截法的函数为：Secant
>
> 功能：用一般弦截法求方程的一个根
>
> 调用格式：root= Secant (f,a,b,eps)
>
> 其中，f：方程名；
>
> a：区间左端点；
>
> b：区间右端点；
>
> eps：根的精度；
>
> root：求得的根。

一般弦截法的 MATLAB 程序代码如下所示：

```
function root=Secant(f,a,b,eps)
%方程表达式：f
%区间左端点：a
%区间右端点：b
%根的精度：eps
%求得的根：root
if(nargin==3)
   eps=1.0e-4;
end
```

```
f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
    root=a;
end
if(f2==0)
    root=b;
end
if(f1*f2>0)
    disp('两端点函数值乘积大于0!');
    return;
else
    tol=1;
    fa=subs(sym(f),findsym(sym(f)),a);
    fb=subs(sym(f),findsym(sym(f)),b);
    root=a-(b-a)*fa/(fb-fa);                        %迭代初始值
    while(tol>eps)
        r1=root;
        fx=subs(sym(f),findsym(sym(f)),r1);
        s=fx*fa;
        if(s==0)
            root=r1;
        else
            if(s>0)
                root=b-(r1-b)*fb/(fx-fb);           %用递推公式2
            else
                root=a-(r1-a)*fa/(fx-fa);           %用递推公式1
            end
        end
        tol=abs(root-r1);
    end
end
```

例 9-8 弦截法求根应用实例。弦截法求方程 $\lg x+\sqrt{x}=2$ 在区间[1,4]上的一个根。

解： 在 MATLAB 命令窗口中输入：

```
>> r=Secant('sqrt(x)+log(x)-2',1,4)
```

输出计算结果为：

```
r =    1.8773
```

由计算结果可知，$\lg x+\sqrt{x}=2$ 在区间[1,4]上的一个根为 $x=1.8773$。

9.2.5.2 单点弦截法

单点弦截法的迭代公式如下：

$$x_{k+1}=x_k-\frac{x_k-a}{f(x_k)-f(a)}f(x_k)$$

在 MATLAB 中编程实现的单点弦截法的函数为：SinleSecant

功能：用单点弦截法求方程的一个根
调用格式：root= SinleSecant (f,a,b,eps)
其中，f：方程名；
　　a：区间左端点；
　　b：区间右端点；
　　eps：根的精度；
　　root：求得的根。

单点弦截法的 MATLAB 程序代码如下所示：

```
function root=SinleSecant(f,a,b,eps)
%方程表达式：f
%区间左端点：a
%区间右端点：b
%根的精度：eps
%求得的根：root
if(nargin==3)
   eps=1.0e-4;
end
f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
   root=a;
end
if(f2==0)
   root=b;
end
if(f1*f2>0)
   disp('两端点函数值乘积大于0!');
   return;
else
   tol=1;
   fa=subs(sym(f),findsym(sym(f)),a);
   fb=subs(sym(f),findsym(sym(f)),b);
   root=b;                          %迭代初始值
   while(tol>eps)
      r1=root;
      fx=subs(sym(f),findsym(sym(f)),r1);
      s=fx*fa;
      if(s==0)
         root=r1;
      else
         root=r1-(r1-a)*fx/(fx-fa);
      end
      tol=abs(root-r1);
   end
end
```

例 9-9 单点弦截法求根应用实例。单点弦截法求方程 $\lg x+\sqrt{x}=2$ 在区间[1,4]上的一个根。

解： 在 MATLAB 命令窗口中输入：

```
>> r=SinleSecant('sqrt(x)+log(x)-2',1,4)
```

输出计算结果为：

```
r =    1.8773
```

由计算结果可知，$\lg x+\sqrt{x}=2$ 在区间[1,4]上的一个根为 $x=1.8773$。

9.2.5.3 双点弦截法

双点弦截法的迭代公式如下：

$$x_{k+1}=x_k-\frac{x_k-x_{k-1}}{f(x_k)-f(x_{k-1})}f(x_k)$$

在 MATLAB 中编程实现的双点弦截法的函数为：DblSecant

功能：用双点弦截法求方程的一个根

调用格式：root= DblSecant (f,a,b,eps)

其中，f：方程名；

a：区间左端点；

b：区间右端点；

eps：根的精度；

root：求得的根。

双点弦截法的 MATLAB 程序代码如下所示：

```
function root=DblSecant(f,a,b,eps)
%方程表达式：f
%区间左端点：a
%区间右端点：b
%根的精度：eps
%求得的根：root
if(nargin==3)
    eps=1.0e-4;
end
f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
    root=a;
end
if(f2==0)
    root=b;
end
if(f1*f2>0)
    disp('两端点函数值乘积大于 0!');
```

```
        return;
    else
        tol=1;
        root=b;
        r2=a;                           %迭代初始值
        while(tol>eps)
            r1 = r2;
            r2 = root;
            f1=subs(sym(f),findsym(sym(f)),r1);
            f2=subs(sym(f),findsym(sym(f)),r2);
            if(f1==0)
                root=r1;
            else
                if(f2==0)
                    root = r2;
                else
                    root=r2-(r2-r1)*f2/(f2-f1);
                end
            end
            tol=abs(root-r2);
        end
    end
```

例 9-10 双点弦截法求根应用实例。双点弦截法求方程 $\lg x+\sqrt{x}=2$ 在区间[1,4]上的一个根。

解：.在 MATLAB 命令窗口中输入：

```
>> r=DblSecant('sqrt(x)+log(x)-2',1,4)
```

输出计算结果为：

```
r =    1.8773
```

由计算结果可知，$\lg x+\sqrt{x}=2$ 在区间[1,4]上的一个根为 $x=1.8773$。

9.2.5.4 平行弦截法

平行弦截法的迭代公式如下：

$$x_{k+1}=x_k-\frac{b-a}{f(b)-f(a)}f(x_k)$$

其中迭代初始值为 x_0。在下面编制的程序中，迭代初始值取为求解区间的左端点。

在 MATLAB 中编程实现的平行弦截法的函数为：PallSecant

功能：用平行弦截法求方程的一个根

调用格式：root= PallSecant (f,a,b,eps)

其中，f：方程名；

a：区间左端点；

b：区间右端点；

eps：根的精度；
root：求得的根。

平行弦截法的 MATLAB 程序代码如下所示：

```
function root= PallSecant (f,a,b,eps)
%方程表达式：f
%区间左端点：a
%区间右端点：b
%根的精度：eps
%求得的根：root
format long;
if(nargin==3)
    eps=1.0e-4;
end
f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
    root=a;
end
if(f2==0)
    root=b;
end
if(f1*f2>0)
    disp('两端点函数值乘积大于 0!');
    return;
else
    tol=1;
    root=a;                              %迭代初始值
    c = (b-a)/(f2-f1);
    while(tol>eps)
        r1 = root;
        fx=subs(sym(f),findsym(sym(f)),r1);
        if(f1==0)
            root=r1;
        else
            root=r1-c*fx;
        end
        tol=abs(root-r1);
    end
end
format short;
```

例 9-11 平行弦截法求根应用实例。平行弦截法求方程 $\lg x+\sqrt{x}=2$ 在区间[1,4]上的一个根。

解：在 MATLAB 命令窗口中输入：

```
>> r=PallSecant('sqrt(x)+log(x)-2',1,4)
```

输出计算结果为:

```
r =    1.8773
```

由计算结果可知，$\lg x+\sqrt{x}=2$ 在区间[1,4]上的一个根为 $x=1.8773$。

9.2.5.5 改进弦截法

改进弦截法的迭代公式如下:

$$x_{k+1}=x_k-\frac{x_k-x_{k-1}}{f(x_k)-f(x_{k-1})}f(x_k)$$

在下一步迭代之前，分以下两种情况:

- 如果 $f(x_k)f(x_{k+1})<0$，则用 $(x_k,f(x_k))$ 代替 $(x_{k-1},f(x_{k-1}))$，用 $(x_{k+1},f(x_{k+1}))$ 代替 $(x_k,f(x_k))$；
- 如果 $f(x_k)f(x_{k+1})>0$，则用 $(x_{k+1},f(x_{k+1}))$ 代替 $(x_k,f(x_k))$，用 $(x_{k-1},\upsilon f(x_{k-1}))d$ 代替 $(x_{k-1},f(x_{k-1}))$，

其中:

$$\upsilon=\begin{cases}0.5 & \text{伊利诺斯法}\\ \dfrac{f(x_k)}{f(x_k)+f(x_{k+1})} & \text{佩加苏法}\\ \upsilon^* & \text{安德森法}\end{cases}$$

且

$$\upsilon^*=\begin{cases}0.5 & \text{当}\ \dfrac{[f(x_{k+1})-f(x_k)](x_k-x_{k-1})}{(x_{k+1}-x_k)[f(x_k)-f(x_{k-1})]}\leqslant 0\\ \dfrac{[f(x_{k+1})-f(x_k)](x_k-x_{k-1})}{(x_{k+1}-x_k)[f(x_k)-f(x_{k-1})]} & \text{当}\ \dfrac{[f(x_{k+1})-f(x_k)](x_k-x_{k-1})}{(x_{k+1}-x_k)[f(x_k)-f(x_{k-1})]}>0\end{cases}$$

在 MATLAB 中编程实现的改进弦截法的函数为: ModifSecant

功能: 用改进弦截法求方程的一个根

调用格式: root= ModifSecant (f,a,b,eps)

其中，f: 方程名;

a: 区间左端点;

b: 区间右端点;

eps: 根的精度;

root: 求得的根。

改进弦截法的 MATLAB 程序代码如下所示:

```
function root=ModifSecant(f,a,b,eps)
%方程表达式: f
%区间左端点: a
%区间右端点: b
%根的精度: eps
```

```
%求得的根: root
format long;
if(nargin==3)
    eps=1.0e-4;
end
f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
    root=a;
end
if(f2==0)
    root=b;
end
if(f1*f2>0)
    disp('两端点函数值乘积大于 0!');
    return;
else
    tol = 1;
    r1 = a;                          %迭代初始值
    r2 = b;
    fv = subs(sym(f),findsym(sym(f)),a);
    while(tol>eps)
        f2=subs(sym(f),findsym(sym(f)),r2);
        root=r2-(r2-r1)*f2/(f2-fv);
        fr=subs(sym(f),findsym(sym(f)),root);
        if(f2*fr<0)
            tol=abs(root-r2);
            r1 = r2;
            r2 = root;
            fv = subs(sym(f),findsym(sym(f)),r1);
        else
            tol=abs(root-r2);
            r2 = root;
            fv = 0.5*subs(sym(f),findsym(sym(f)),r1);     %伊利诺斯法
        end
    end
end
format short;
```

例 9-12　改进弦截法求根应用实例。改进弦截法求方程 $\lg x+\sqrt{x}=2$ 在区间[1,4]上的一个根。

解： 在 MATLAB 命令窗口中输入:

```
>> r=ModifSecant('sqrt(x)+log(x)-2',1,4)
```

输出计算结果为:

```
r =    1.8773
```

由计算结果可知，$\lg x+\sqrt{x}=2$ 在区间[1,4]上的一个根为 $x=1.8773$。从效率方面讲，改进弦截法比前几种弦截法的速度都要快。

9.2.6 史蒂芬森法

史蒂芬森法是弦截法的一种变形，它的递推公式为：

$$x_k = x_{k-1} - \frac{f(x_{k-1})}{f(x_{k-1} + f(x_{k-1})) - f(x_{k-1})} f(x_{k-1})，且有 x_1 = a - \frac{f(a)}{f(a + f(a)) - f(a)} f(a)。$$

史蒂芬森法的收敛速度也很快。

> 在 MATLAB 中编程实现的史蒂芬森法的函数为：StevenSecant
>
> 功能：用史蒂芬森法求方程的一个根
>
> 调用格式：root= StevenSecant (f,a,b,eps)
>
> 其中，f：方程名；
>
> a：区间左端点；
>
> b：区间右端点；
>
> eps：根的精度；
>
> root：求得的根。

史蒂芬森弦截法的 MATLAB 程序代码如下所示：

```
function root=StevenSecant(f,a,b,eps)
%方程表达式：f
%区间左端点：a
%区间右端点：b
%根的精度：eps
%求得的根：root
if(nargin==3)
    eps=1.0e-4;
end
f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
    root=a;
end
if(f2==0)
    root=b;
end
if(f1*f2>0)
    disp('两端点函数值乘积大于0!');
    return;
else
    tol=1;
    fa=subs(sym(f),findsym(sym(f)),a);
    fb=subs(sym(f),findsym(sym(f)),b);
    faa=subs(sym(f),findsym(sym(f)),fa+a);
    root=a-fa*fa/(faa-fa);                              %迭代初始值
    while(tol>eps)
        r1=root;
```

```
        fx=subs(sym(f),findsym(sym(f)),r1);
        v=fx+r1;
        fxx=subs(sym(f),findsym(sym(f)),v);
        root=r1-fx*fx/(fxx-fx);                         %递推的核心公式
        tol=abs(root-r1);
    end
end
```

例 9-13 史蒂芬森弦截法求根应用实例。史蒂芬森弦截法求方程 $\lg x+\sqrt{x}=2$ 在区间[1,4]上的一个根。

解： 在 MATLAB 命令窗口中输入：

```
>> r=StevenSecant('sqrt(x)+log(x)-2',1,4)
Warning: Log of zero.
> In sym.subs at 127
  In StevenSecant at 22
Warning: Log of zero.
> In sym.subs at 127
  In StevenSecant at 28
r =     1
```

结果出现错误，改正方法之一是把求解区间缩短为[1.5,2]：

```
>> r=StevenSecant('sqrt(x)+log(x)-2',1.5,2)
r =    1.8773
```

这样就得出了正确的根 1.8773。从上面的例子可以看出求解区间的选择也是很重要的，一般来说，求解区间越短越好。

9.2.7 劈因子法

劈因子法通过找到实系数代数方程

$$f(x)=x^n+a_1x^{n-1}+\cdots+a_{n-1}x+a_n=0$$

的一个二次因子 $w(x)=x^2+px+q$，从而找到方程的两个实根或一对共轭复根。

求 $w(x)$ 的算法如下：

❶ 任意选取 $w(x)$ 的一个近似值，令 $w(x)=x^2+px+q$

- 令 $f(x)=w(x)Q(x)+R(x)=(x^2+px+q)(x^{n-2}+b_1x^{n-3}+\cdots+b_{n-3}x+b_{n-2})+(r_1x+r_2)$

  其中

$$\begin{cases}b_k=a_k-pb_{k-1}-qb_{k-2}(k=1,2,\cdots,n)\\ b_{-1}=0,b_0=1\\ r_1=a_{n-1}-pb_{n-2}-qb_{n-3}\\ r_2=a_n-qb_{n-2}\end{cases}$$

- 令 $xQ(x)=w(x)M(x)+R_1(x)$

$$=(x^2+px+q)(x^{n-3}+c_1x^{n-4}+\cdots+c_{n-4}x+c_{n-3})+(R_{11}x+R_{12})$$

其中

$$\begin{cases} c_k = b_k - pc_{k-1} - qc_{k-2} (k = 1, 2, \cdots, n-3) \\ c_{-1} = 0, c_0 = 1 \\ R_{11} = b_{n-2} - pc_{n-3} - qc_{n-4} \\ R_{12} = -qc_{n-3} \end{cases}$$

❷ 解二元一次方程组：

$$\begin{cases} R_{11}u + R_{21}v = r_1 \\ R_{22}u + R_{12}v = r_2 \end{cases}$$

其中

$$R_{21} = b_{n-3} - pc_{n-4} - qc_{n-5}$$
$$R_{22} = b_{n-2} - qc_{n-4}$$

❸ 修正 $w(x)$：

$$w(x) = x^2 + (p+u)x + (q+v)$$

如此直到满足一定的精度为止。

在 MATLAB 中编程实现的劈因子法的函数为：PYZ

功能：用劈因子法求方程的一个二次因子

调用格式：[p,q]=PYZ(a,M,p0,q0)

其中，a：方程系数；

M：迭代步数；

p0：初始一次项系数；

q0：初始常数项；

p：求得的二次因子的一次项系数；

q：求得的二次因子的常数项。

劈因子法的 MATLAB 程序代码如下所示：

```
function [p,q]=PYZ(a,M,p0,q0)
%方程系数：a
%迭代步数：M
%初始一次项系数：b
%初始常数项：eps
%求得的二次因子的一次项系数：p
%求得的二次因子的常数项：q
format long;
n = length(a);
b = zeros(n,1);
c = zeros(n-1,1);
p = p0;
q = q0;
for k=1:M
    b(1) = 0;
```

```
    b(2) = 1;
    for i=3:n
       b(i) = a(i-2) - p*b(i-1)-q*b(i-2);
    end
    r1 = a(n-1) - p*b(n)-q*b(n-1);
    r2 = a(n) - q*b(n);
    c(1) = 0;
    c(2) = 1;
    for i=3:n-1
       c(i) = b(i) - p*c(i-1)-q*c(i-2);
    end
    R11 = b(n) - p*c(n-1)-q*c(n-2);
    R12 = -q*c(n-1);
    R21 = b(n-1) - p*c(n-2)-q*c(n-3);
    R22 = b(n)-q*c(n-2);
    A = [R11 R21;R22 R12];
    B = [r1;r2];
    u = A\B;
    p = p+u(1);
    q = q+u(2);
end
format short;
```

例 9-14 劈因子法求根应用实例。求 $y=x(x+1)^2(x-1)(x+3)=x^5+4x^4+2x^3-4x^2-3x$ 的一个二次因子。

解： 在 MATLAB 命令窗口中输入：

```
>> a=[4 2 -4 -3 0];
>> [p,q] = PYZ(a,20,3,8)
p =    2.0000
q =    1.0000
>> [p,q] = PYZ(a,50,2,5)
p =    2.0000
q =   -3.0000
```

上面的结果中第一次求出的二次因子为 $x^2+2x+1=(x+1)^2$，第二次求出的二次因子为 $x^2+2x-3=(x-1)(x+3)$。初始值的选取很重要，如果选得不好，可能就求不出二次因子。

9.2.8 抛物线法

弦截法其实是用不断缩短的直线来近似函数 $f(x)$，而抛物线法采用三个点来近似函数 $f(x)$，并且把抛物线与横轴的交点来逼近函数 $f(x)$ 的根。它的算法过程介绍如下。

❶ 选定初始值 x_0、x_1、x_2，并计算 $f(x_0)$、$f(x_1)$、$f(x_2)$ 和以下差分：

$$f(x_2,x_1)=\frac{f(x_2)-f(x_1)}{x_2-x_1}$$

$$f(x_1,x_0)=\frac{f(x_1)-f(x_0)}{x_1-x_0}$$

$$f(x_2,x_1,x_0)=\frac{f(x_2,x_1)-f(x_1,x_0)}{x_2-x_0}$$

一般取 $x_0=a$ ， $x_1=b$ ， $a<x_2<b$ 。注意不要使三点共线。

❷ 用牛顿插值法对三点 $(x_0,f(x_0))$ 、 $(x_1,f(x_1))$ 、 $(x_2,f(x_2))$ 进行插值得到一条抛物线，它有两个根：

$$x_3=x_2+\frac{-B\pm\sqrt{B^2-4AC}}{2C}$$

其中，

$$A=f(x_2),C=f(x_2,x_1,x_0)$$

$$B=f(x_2,x_1)+f(x_2,x_1,x_0)(x_2-x_1)$$

两个根中只取靠近 x_2 的那个根，即±号取与 B 同号。即

$$x_3=x_2-\frac{2A}{B+\mathrm{sgn}(B)\sqrt{B^2-4AC}}$$

❸ 用 x_1 、 x_2 、 x_3 代替 x_0 、 x_1 、 x_2 ，重复以上步骤。并有以下递推公式：

$$x_{n+1}=x_n-\frac{2A_n}{B_n+\mathrm{sgn}(B_n)\sqrt{{B_n}^2-4A_nC_n}}$$

其中，

$$A_n=f(x_n),C_n=f(x_n,x_{n-1},x_{n-2})$$

$$B_n=f(x_n,x_{n-1})+f(x_n,x_{n-1},x_{n-2})(x_n-x_{n-1})$$

❹ 进行精度控制。

在 MATLAB 中编程实现的抛物线法的函数为：Parabola

功能：用抛物线法求方程的一个根

调用格式：root= Parabola (f,a,b,x,eps)

其中，f：方程名；

a：区间左端点；

b：区间右端点；

x：迭代初始点；

eps：根的精度；

root：求得的根。

抛物线法的 MATLAB 程序代码如下所示：

```
function root=Parabola(f,a,b,x,eps)
%方程表达式：f
%区间左端点：a
%区间右端点：b
%迭代初始点：x
%根的精度：eps
```

```
%求得的根: root
if(nargin==4)
    eps=1.0e-4;
end
f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
    root=a;
end
if(f2==0)
    root=b;
end
if(f1*f2>0)
    disp('两端点函数值乘积大于 0!');
    return;
else
    tol=1;
    fa=subs(sym(f),findsym(sym(f)),a);
    fb=subs(sym(f),findsym(sym(f)),b);
    fx=subs(sym(f),findsym(sym(f)),x);
    d1=(fb-fa)/(b-a);
    d2=(fx-fb)/(x-b);
    d3=(d2-d1)/(x-a);
    B=d2+d3*(x-b);
    root=x-2*fx/(B+sign(B)*sqrt(B^2-4*fx*d3));
    t=zeros(3);
    t(1)=a;
    t(2)=b;
    t(3)=x;
    while(tol>eps)
        t(1)=t(2);                                      %保存三个点
        t(2)=t(3);
        t(3)=root;
        f1=subs(sym(f),findsym(sym(f)),t(1));           %计算三点的函数值
        f2=subs(sym(f),findsym(sym(f)),t(2));
        f3=subs(sym(f),findsym(sym(f)),t(3));
        d1=(f2-f1)/(t(2)-t(1));                         %计算三个差分
        d2=(f3-f2)/(t(3)-t(2));
        d3=(d2-d1)/(t(3)-t(1));
        B=d2+d3*(t(3)-t(2));                            %计算算法中的 B
        root=t(3)-2*f3/(B+sign(B)*sqrt(B^2-4*f3*d3));
        tol=abs(root-t(3));
    end
end
```

例 9-15　抛物线法求根应用实例。采用抛物线法求方程 $\frac{1}{\sqrt{x}}+x-2=0$ 的一个根，求解区间分别为[0.5,1.5]和[0.1,1]。

解： 在 MATLAB 命令窗口中输入：

```
>> r=Parabola('1/sqrt(x)+x-2',0.5,1.5,0.8) %求解区间[0.5,1.5]，迭代初值为 0.8
```

```
r =     1
>> r=Parabola('1/sqrt(x)+x-2',0.1,1,0.5) %求解区间[0.1,1]，迭代初值为 0.5
r =   0.3820
```

由此可以看出抛物线法也能求出方程 $\frac{1}{\sqrt{x}}+x-2=0$ 的两个根来。

9.2.9 钱伯斯法

钱伯斯法和抛物线法一样，也是三点迭代公式，不过它有两个迭代公式，其一为：

$$x_{n+1}=\frac{x_{n-2}y_{n-1}y_n}{(y_{n-2}-y_{n-1})(y_{n-2}-y_n)}+\frac{x_{n-1}y_{n-2}y_n}{(y_{n-1}-y_{n-2})(y_{n-1}-y_n)}+\frac{x_n y_{n-1}y_{n-2}}{(y_n-y_{n-1})(y_n-y_{n-2})}$$

另一个为：

$$x_{n+1}=\frac{x_n^* y_{n-1}y_n}{(y_n^*-y_{n-1})(y_n^*-y_n)}+\frac{x_{n-1}y_n^* y_n}{(y_{n-1}-y_n^*)(y_{n-1}-y_n)}+\frac{x_n y_{n-1}y_n^*}{(y_n-y_{n-1})(y_n-y_n^*)}$$

其中：

$$x_n^*=\begin{cases}\dfrac{x_{n-1}+x_n}{2}\\ x_n-\dfrac{x_n-x_{n-1}}{y_n-y_{n-1}}\end{cases}$$

在 MATLAB 中编程实现的钱伯斯法的函数为：QBS

功能：用钱伯斯法求方程的一个根

调用格式：root= QBS (f,a,b,x,eps)

其中，f：方程名；

a：区间左端点；

b：区间右端点；

x：迭代初始点；

eps：根的精度；

root：求得的根。

钱伯斯法的 MATLAB 程序代码如下所示：

```
function root=QBS(f,a,b,x,eps)
%方程表达式：f
%区间左端点：a
%区间右端点：b
%迭代初始点：x
%根的精度：eps
%求得的根：root
format long;
if(nargin==4)
   eps=1.0e-6;
end
```

```
fa=subs(sym(f),findsym(sym(f)),a);
fb=subs(sym(f),findsym(sym(f)),b);
fx=subs(sym(f),findsym(sym(f)),x);
if(fa==0)
    root=a;
    return;
end
if(fb==0)
    root=b;
    return;
end
if(fx == 0)
    root = x;
    return;
else
    tol=1;
    t=zeros(3);
    t(1)=a;
    t(2)=x;
    t(3)=b;
    while(tol>eps)
        f1=subs(sym(f),findsym(sym(f)),t(1));          %计算三点的函数值
        f2=subs(sym(f),findsym(sym(f)),t(2));
        f3=subs(sym(f),findsym(sym(f)),t(3));
        d1=f2-f1;
        d2=f3-f2;
        d3=f3-f1;
        if (d1 == 0 || d2 == 0 || d3 == 0)
            disp('除0错误，请重新选择三个初始点！');
            root = NaN;
            return;
        end
        root=t(1)*f2*f3/d1/d3+t(2)*f1*f3/d1/d2+t(3)*f2*f1/d2/d3;
        tol=abs(root-t(3));
        ft = subs(sym(f),findsym(sym(f)),root);
        if( f2*ft < 0)
            if root > t(2)
                t(1)=t(2);
                t(2)=root;
            else
                t(3) = t(2);
                t(2) = root;
            end
        else
            if f2 < 0
                if root > t(2)
                    t(1)=t(2);
                    t(2)=root;
                else
                    t(1) = root;
                end
```

```
            else
                if root > t(2)
                    t(3) = root;
                else
                    t(3) = t(2);
                    t(2) = root;
                end
            end
        end
    end
end
format short;
```

例 9-16　钱伯斯法求根应用实例。钱伯斯法求方程 $x^2-3x+2=0$ 在区间[0,1.8]的一个根。

解： 在 MATLAB 命令窗口中输入：

```
>> syms x;
>> y = x^2-3*x+2;
>> r=QBS(y,0,1.8,0.6) %求解区间[0,1.8]，迭代初值为 0.6
r =    1.0000
```

由此求出方程 $x^2-3x+2=0$ 的一个根为 $x=1$。

9.2.10 牛顿法

弦截法本质上是一种割线法，它从两端向中间逐渐逼近方程的根；牛顿法本质上是一种切线法，它从一端向一个方向逼近方程的根，其递推公式为：

$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}$$

初始值可以取 $f'(a)$ 和 $f'(b)$ 较大者，这样可以加快收敛速度。

和牛顿法有关的还有简化牛顿法和牛顿下山法。

在 MATLAB 中编程实现的牛顿法的函数为：NewtonRoot

功能：用牛顿法求方程的一个根

调用格式：root= NewtonRoot (f,a,b,eps)

其中，f：方程名；

a：区间左端点；

b：区间右端点；

eps：根的精度；

root：求得的根。

牛顿法的 MATLAB 程序代码如下所示：

```
function root=NewtonRoot(f,a,b,eps)
%方程表达式：f
```

```
%区间左端点: a
%区间右端点: b
%根的精度: eps
%求得的根: root
if(nargin==3)
    eps=1.0e-4;
end
f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
    root=a;
end
if(f2==0)
    root=b;
end
if(f1*f2>0)
    disp('两端点函数值乘积大于 0!');
    return;
else
    tol=1;
    fun=diff(sym(f));                                    %求导数
    fa=subs(sym(f),findsym(sym(f)),a);
    fb=subs(sym(f),findsym(sym(f)),b);
    dfa=subs(sym(fun),findsym(sym(fun)),a);
    dfb=subs(sym(fun),findsym(sym(fun)),b);
    if(dfa>dfb)                                          %初始值取两端点导数较大者
        root=a-fa/dfa;
    else
        root=b-fb/dfb;
    end
    while(tol>eps)
        r1=root;
        fx=subs(sym(f),findsym(sym(f)),r1);
        dfx=subs(sym(fun),findsym(sym(fun)),r1);        %求该点的导数值
        root=r1-fx/dfx;                                  %迭代的核心公式
        tol=abs(root-r1);
    end
end
```

例 9-17 牛顿法求根应用实例。牛顿法求方程 $\sqrt{x}-x^3+2=0$ 在区间[0,5.2]的一个根。

解: 在 MATLAB 命令窗口中输入:

```
>> r=NewtonRoot('sqrt(x)-x^3+2',0.5,2)
```

输出计算结果为:

```
r =    1.4759
```

由计算结果可知，$\sqrt{x}-x^3+2=0$ 的一个根为 $x=1.4759$。

需要注意的是，初始值的选择不要使得其导数为 0。

9.2.10.1 简化牛顿法

将牛顿法的递推公式改为：

$$x_{n+1} = x_n - \lambda f(x_n)$$

为保证收敛，系数λ只需满足$0 < \lambda < \dfrac{2}{f'(x_n)}$即可。如果$\lambda$取常数$\dfrac{1}{f'(x_0)}$，则称为简化牛顿法。

> 在 MATLAB 中编程实现的简化牛顿法的函数为：SimpleNewton
>
> 功能：用简化牛顿法求方程的一个根
>
> 调用格式：root= SimpleNewton (f,a,b,eps)
>
> 其中，f：方程名；
>
> a：区间左端点；
>
> b：区间右端点；
>
> eps：根的精度；
>
> root：求得的根。

简化牛顿法的 MATLAB 代码如下所示：

```
function root=SimpleNewton(f,a,b,eps)
%方程表达式：f
%区间左端点：a
%区间右端点：b
%根的精度：eps
%求得的根：root
if(nargin==3)
    eps=1.0e-4;
end
f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
    root=a;
end
if(f2==0)
    root=b;
end
if(f1*f2>0)
    disp('两端点函数值乘积大于 0!');
    return;
else
    tol=1;
    fun=diff(sym(f));                          %求导数
    fa=subs(sym(f),findsym(sym(f)),a);
    fb=subs(sym(f),findsym(sym(f)),b);
    dfa=subs(sym(fun),findsym(sym(fun)),a);
```

```
    dfb=subs(sym(fun),findsym(sym(fun)),b);
    if(dfa>dfb)                                   %初始值取两端点导数较大者
        df0=1/dfa;
        root=a-df0*fa;
    else
        df0=1/dfb;
        root=b-df0*fb;
    end
    while(tol>eps)
        r1=root;
        fx=subs(sym(f),findsym(sym(f)),r1);
        root=r1-df0*fx;                           %迭代的核心公式，其中 df0 为常数
        tol=abs(root-r1);
    end
end
```

例 9-18 简化牛顿法求根应用实例。简化牛顿法求方程 $\sqrt{x}-x^3+2=0$ 在区间[1.2,2]的一个根。

解： 在 MATLAB 命令窗口中输入：

```
>> r=SimpleNewtonRoot('sqrt(x)-x^3+2',1.2,2)
```

输出计算结果为：

```
r =    1.4759
```

由计算结果可知，$\sqrt{x}-x^3+2=0$ 的一个根为 $x=1.4759$。

9.2.10.2 牛顿下山法

牛顿下山法收敛速度很快，很容易得出方程的根。牛顿下山法的递推公式为：

$$x_{n+1}=x_n-\alpha\frac{f(x_n)}{f'(x_n)}(0<\alpha\leqslant 1)$$

α 称为下山因子，它的确定方法如下：

1. 先取 $\alpha=1$，求出 x_{n+1}；
2. 判断条件 $|f(x_{n+1})|<|f(x_n)|$ 是否满足；
3. 不满足的话令 $\alpha=0.5\alpha$；
4. 再验证直到 $|f(x_{n+1})|<|f(x_n)|$。

> 在 MATLAB 中编程实现的牛顿下山法的函数为：NewtonDown
> 功能：用牛顿下山法求方程的一个根
> 调用格式：root= NewtonDown (f,a,b,eps)
> 其中，f：方程名；
> a：区间左端点；
> b：区间右端点；
> eps：根的精度；

root：求得的根。

牛顿下山法的 MATLAB 程序代码如下所示：

```
function root=NewtonDown(f,a,b,eps)
%方程表达式：f
%区间左端点：a
%区间右端点：b
%根的精度：eps
%求得的根：root
if(nargin==3)
    eps=1.0e-4;
end
f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
    root=a;
end
if(f2==0)
    root=b;
end
if(f1*f2>0)
    disp('两端点函数值乘积大于 0!');
    return;
else
    tol=1;
    fun=diff(sym(f));
    fa=subs(sym(f),findsym(sym(f)),a);
    fb=subs(sym(f),findsym(sym(f)),b);
    dfa=subs(sym(fun),findsym(sym(fun)),a);
    dfb=subs(sym(fun),findsym(sym(fun)),b);
    if(dfa>dfb)                                    %迭代初始值的选取
        root=a;
    else
        root=b;
    end
    while(tol>eps)
        r1=root;
        fx=subs(sym(f),findsym(sym(f)),r1);
        dfx=subs(sym(fun),findsym(sym(fun)),r1);
        toldf=1;
        alpha=2;
        while toldf>0                              %此循环为下山因子的选取过程
            alpha=alpha/2;
            root=r1-alpha*fx/dfx;                  %迭代的核心公式
            fv=subs(sym(f),findsym(sym(f)),root);
            toldf=abs(fv)-abs(fx);
        end
        tol=abs(root-r1);
    end
```

```
end
```

例 9-19 牛顿下山法求根应用实例。牛顿下山法求方程 $\sqrt{x}-x^3+2=0$ 在区间[1,2]的一个根。

解： 在 MATLAB 命令窗口中输入：

```
>> r=NewtonDown('sqrt(x)-x^3+2',1,2)
```

输出计算结果为：

```
r =    1.4759
```

由计算结果可知，$\sqrt{x}-x^3+2=0$ 的一个根为 $x=1.4759$。

9.2.11 逐次压缩牛顿法

逐次压缩牛顿法可用于求实多项式

$$f(x)=x^n+a_1x^{n-1}+\cdots+a_{n-1}x+a_n$$

的全部实根。

❶ 用贝努利法求出根的上下界；

❷ 用牛顿法求 $f(x)$ 的一个根 x_0，令 $f(x)=(x-x_0)F_1(x)$

其中

$$\begin{cases} F_1(x)=b_0x^{n-1}+b_1x^{n-2}+\cdots+b_{n-2}x+b_{n-1} \\ b_0=1 \\ b_k=a_k+x_0b_{k-1}(k=1,2,\cdots,n-1) \end{cases}$$

❸ 用牛顿法求 $F_1(x)$ 的一个根 x_1，令 $F_2(x)=(x-x_1)F_2(x)$

其中

$$\begin{cases} F_2(x)=c_0x^{n-2}+c_1x^{n-3}+\cdots+c_{n-3}x+c_{n-2} \\ c_0=1 \\ c_k=b_k+x_1c_{k-1}(k=1,2,\cdots,n-2) \end{cases}$$

如此继续下去，可逐步求出 $f(x)$ 的全部实根。

> 在 MATLAB 中编程实现的逐次压缩牛顿法的函数为：YSNewton
> 功能：逐次压缩牛顿法求多项式的全部实根
> 调用格式：x=YSNewton(a)
> 其中，a：方程系数；
> 　　　x：求得的所有实根。

贝努利法求按模最大实根的 MATLAB 程序代码如下所示：

```
function x=YSNewton(a)
%方程系数：a
%求得的所有实根：x
format long;
```

```
n = length(a);
max_X = BenvliMAX(a);
b = zeros(n,1);
x = zeros(n,1);
syms t real;
f = power(t,n);
m = 0:n-1;
s = power(t,m);
f = f + dot(s, a(n:-1:1));
for k=1:n
    x(k) = NewtonRoot(f,-max_X,max_X);
    b = 0;
    b(1) = 1;
    for i=2:n-k+1
        b(i) = a(i-1) + x(k)*b(i-1);
    end
    m = 0:n-k;
    s = power(t,m);
    f = dot(s, b(n-k+1:-1:1));
    a = b(2:length(b));
end
format short;
```

例 9-20　逐次压缩牛顿法求根应用实例。逐次压缩牛顿法求方程 $x^4+4x^3+2x^2-4x-3=0$ 的全部实根。

解： 在 MATLAB 命令窗口中输入：

```
>> a=[4 2 -4 -3];
>> s = YSNewton(a)
s =    1.0000
      -0.9999
      -1.0001
      -3.0000
```

由此求出方程

$$x^4+4x^3+2x^2-4x-3=(x-1)(x+1)^2(x+3)=0$$

的全部实根为：

$$\begin{cases} x_1=1.0000 \\ x_2=-0.999 \\ x_3=-1.0001 \\ x_4=-3.0000 \end{cases}$$

9.2.12　联合法

联合法是将牛顿法和弦截法结合起来的方法，迭代公式有两种，第一种迭代公式为：

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$$

$$x_n^* = x_{n-1}^* - \frac{(x_{n-1} - x_{n-1}^*)}{f(x_{n-1}) - f(x_{n-1}^*)} f(x_{n-1}^*)$$

其初始值为：

$$x_1 = a - \frac{f(a)}{f'(a)}$$

$$x_1^* = b - \frac{a-b}{f(a)-f(b)} f(b)$$

在 MATLAB 中编程实现的联合法 1 的函数为：Union1

功能：用联合法 1 求方程的一个根

调用格式：root= Union1 (f,a,b,eps)

其中，f：方程名；

a：区间左端点；

b：区间右端点；

eps：根的精度；

root：求得的根。

联合法 1 的 MATLAB 代码如下所示：

```
function root = Union1(f,a,b,eps)
%方程表达式：f
%区间左端点：a
%区间右端点：b
%根的精度：eps
%求得的根：root
format long;
if(nargin==3)
   eps=1.0e-6;
end
fa=subs(sym(f),findsym(sym(f)),a);
fb=subs(sym(f),findsym(sym(f)),b);
if(fa==0)
   root=a;
end
if(fb==0)
   root=b;
end
tol=1;
df=diff(sym(f));                              %求导数
dfa=subs(sym(df),findsym(sym(df)),a);
x1 = a - fa/dfa;
x1_s = b - fb*(b-a)/(fb-fa);
while tol>eps
```

```
    fn=subs(sym(f),findsym(sym(f)),x1);
    dfn=subs(sym(df),findsym(sym(df)),x1);
    fn_s=subs(sym(f),findsym(sym(f)),x1_s);
    x2 = x1 - fn/dfn;
    x2_s = x1_s - (x1 - x1_s)*fn_s/(fn - fn_s);
    tol = abs(x2_s - x1_s);
    x1 = x2;
    x1_s = x2_s;
end
root = x2_s;
format short;
```

例 9-21 联合法 1 求根应用实例。采用联合法 1 求方程 $x^3-2x^2+3x-1=0$ 在区间[0,3]的一个根。

解： 在 MATLAB 命令窗口中输入：

```
>> syms x;
>> y = x^3 - 2*x^2+ 3*x-1;
>> r = Union1(y,0, 3)
r =    0.4302
```

由此求出方程 $x^3-2x^2+3x-1=0$ 的一个根为 $x=0.4302$。

联合法的第二种迭代公式为：

$$\begin{cases} x_n = x_{n-1} - \dfrac{(x_{n-1}-x_{n-1}^*)}{f(x_{n-1})-f(x_{n-1}^*)} f(x_{n-1}) \\ x_n^* = x_{n-1}^* - \dfrac{f(x_{n-1}^*)}{f'(x_{n-1}^*)} \end{cases}$$

其初始值为：

$$x_1 = a - \frac{a-b}{f(a)-f(b)} f(a)$$

$$x_1^* = b - \frac{f(b)}{f'(b)}$$

> 在 MATLAB 中编程实现的联合法 2 的函数为：Union2
>
> 功能：用联合法 2 求方程的一个根
>
> 调用格式：root= Union2 (f,a,b,eps)
>
> 其中，f：方程名；
>
> a：区间左端点；
>
> b：区间右端点；
>
> eps：根的精度；
>
> root：求得的根。

联合法 2 的 MATLAB 代码如下所示：

```
function root = Union2(f,a,b,eps)
```

```
%方程表达式：f
%区间左端点：a
%区间右端点：b
%根的精度：eps
%求得的根：root
format long;
if(nargin==3)
    eps=1.0e-4;
end
fa=subs(sym(f),findsym(sym(f)),a);
fb=subs(sym(f),findsym(sym(f)),b);
if(fa==0)
    root=a;
end
if(fb==0)
    root=b;
end
tol=1;
df=diff(sym(f));                                    %求导数
dfb=subs(sym(df),findsym(sym(df)),b);
x1 = a - (b-a)*fa/(fb-fa);
x1_s = b - fb/dfb;
while tol>eps
    fn = subs(sym(f),findsym(sym(f)),x1);
    dfn_s = subs(sym(df),findsym(sym(df)),x1_s);
    fn_s = subs(sym(f),findsym(sym(f)),x1_s);
    x2 = x1 - (x1_s - x1)*fn/(fn_s - fn);
    x2_s = x1_s - fn_s/dfn_s ;
    tol = abs(x2_s - x1_s);
    x1 = x2;
    x1_s = x2_s;
end
root = x2_s;
format short;
```

例 9-22　联合法 2 求根应用实例。采用联合法 2 求方程 $x^3-2x^2+3x-1=0$ 在区间[0,3]的一个根。

解： 在 MATLAB 命令窗口中输入：

```
>> syms x;
>> y = x^3 - 2*x^2+ 3*x-1;
>> r = Union2(y,0, 3)
r =    0.4302
```

由此求出方程 $x^3-2x^2+3x-1=0$ 的一个根为 $x=0.4302$。

9.2.13 两步迭代法

两步迭代法的一般公式为：

$$y_n = g(x_n)$$
$$x_{n+1} = h(y_n)$$

比较常用的有以下两种形式：

$$(1)\begin{cases} y_n = x_n - \dfrac{f(x_n)}{f'(x_n)} \\ x_{n+1} = y_n - \dfrac{f(y_n)}{f'(x_n)} \end{cases}$$

$$(2)\begin{cases} y_n = x_n - \dfrac{f(x_n)}{f'(x_n)} \\ x_{n+1} = y_n - \dfrac{f(y_n)(y_n - x_n)}{2f(y_n) - f(x_n)} \end{cases}$$
。

在 MATLAB 中编程实现的两步迭代法的函数为：TwoStep

功能：用两步迭代法求方程的一个根

调用格式：root=TwoStep(f,a,b,type,eps)

其中，f：方程名；

a：区间左端点；

b：区间右端点；

type：迭代形式；

eps：根的精度；

root：求得的根。

两步迭代法的 MATLAB 代码如下所示：

```
function root=TwoStep(f,a,b,type,eps)
%方程表达式：f
%区间左端点：a
%区间右端点：b
%迭代形式：type;
%根的精度：eps
%求得的根：root
function root=TwoStep(f,a,b,type,eps)
%type=1  第一种形式
%type=2  第二种形式
if(nargin==4)
   eps=1.0e-4;
end
f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
```

```
    root=a;
end
if(f2==0)
    root=b;
end
if(f1*f2>0)
    disp('两端点函数值乘积大于 0!');
    return;
else
    tol=1;
    fun=diff(sym(f));
    fa=subs(sym(f),findsym(sym(f)),a);
    fb=subs(sym(f),findsym(sym(f)),b);
    dfa=subs(sym(fun),findsym(sym(fun)),a);
    dfb=subs(sym(fun),findsym(sym(fun)),b);
    if(dfa>dfb)
        root=a;
    else
        root=b;
    end
    while(tol>eps)
        if(type==1)                                   %第一种形式的两步迭代法
            r1=root;
            fx1=subs(sym(f),findsym(sym(f)),r1);
            dfx=subs(sym(fun),findsym(sym(fun)),r1);
            r2=r1-fx1/dfx;                            %第一步迭代
            fx2=subs(sym(f),findsym(sym(f)),r2);
            root=r2-fx2/dfx;                          %第二步迭代
            tol=abs(root-r1);
        else
            r1=root;
            fx1=subs(sym(f),findsym(sym(f)),r1);
            dfx=subs(sym(fun),findsym(sym(fun)),r1);
            r2=r1-fx1/dfx;                            %第一步迭代
            fx2=subs(sym(f),findsym(sym(f)),r2);
            root=r2-fx2*(r2-r1)/(2*fx2-fx1);          %第二步迭代
            tol=abs(root-r1);
        end
    end
end
```

例 9-23　两步迭代法求根应用实例。利用两步迭代法的两种形式求方程 $\lg x-\sin(x)+1=0$ 在区间[0.1,3]的一个根。

解： 在 MATLAB 命令窗口中输入：

```
>> r=TwoStep('(log(x)-sin(x)+1)',0.1,3,1)
r =    0.7013
>> r=TwoStep('(log(x)-sin(x)+1)',0.1,3,2)
r =    0.7013
```

从例子可以看出，两种形式的两步迭代法得出的结果是一样的。

9.2.14 蒙特卡洛法

蒙特卡洛法求根的算法介绍如下：

❶ 令 $j=0$，任选一个初值 $x^{(0)}$，计算

$$F^{(0)}=f(x^{(0)})$$

❷ 产生 $(-B,B)$ 区间上均匀分布的随机数 r，以 $x^{(0)}+r$ 作为自变量，计算

$$F^{(1)}=f(x^{(1)}+r)$$

❸ 若 $F^{(1)}\geqslant F^{(0)}$，则返回上一步，重新产生随机数，直到找到某个 r'，使得 $F^{(1)}<F^{(0)}$

❹ 若 $\left|F^{(1)}\right|\leqslant\varepsilon$，则 $x^{(0)}+r'$ 为方程的一个根；否则转❷。

在 MATLAB 中编程实现的蒙特卡洛法的函数为：Montecarlo

功能：用蒙特卡洛法求方程的一个根

调用格式：　root = Montecarlo(f,B,x0,eps)

其中，f：方程名；

B：随机数发生区间；

x0：初始值；

eps：根的精度；

root：求得的根。

蒙特卡洛法的 MATLAB 代码如下所示：

```
function root = Montecarlo(f,B,x0,eps)
%方程表达式：f
%随机数发生区间：B
%初始值：x0
%根的精度：eps
%求得的根：root
format long;
if(nargin==3)
    eps=1.0e-4;
end
Fx = subs(sym(f),findsym(sym(f)),x0);
while abs(Fx)>eps
    Fx = subs(sym(f),findsym(sym(f)),x0);
    Fx1 = abs(Fx) + 10;
    while (abs(Fx1) >= abs(Fx))
        r = 2*B*rand()-B;
        x1 = x0 + r;
        Fx1 = subs(sym(f),findsym(sym(f)),x1);
    end
    x0 = x1;
end
```

```
root = x1;
format short;
```

例 9-24 蒙特卡洛法求根应用实例。用蒙特卡洛法求方程 $\ln t+10-t=0$ 的一个根。

解：选取初始区间为[−3,3]，初始迭代值为 10，在 MATLAB 命令窗口中输入：

```
>> syms t;
>> y=log(t)-t+10;
>> r = Montecarlo(y,3,10)
r =   12.5279
```

蒙特卡洛法比较特殊，有时候很快就能得出方程的根，有时候却几十分钟也求不出来，带有很大的随机性。

9.2.15 重根的迭代方法

如果方程存在重根，那么前面的方法都可能不收敛或者收敛速度很慢，因此针对二重根的情况有以下的迭代公式：

$$x_{n+1}=x_n-\frac{f'(x_n)f(x_n)}{[f'(x_n)]^2-f(x_n)f''(x_n)}$$

但它要求计算二阶导数，如果方程复杂点的话，计算量则比较大。

> 在 MATLAB 中编程实现的重根迭代法的函数为：MultiRoot
>
> 功能：求存在重根的方程的一个重根
>
> 调用格式： root=MultiRoot(f,a,b,eps)
>
> 其中，f：方程名；
>
> a：区间左端点；
>
> b：区间右端点；
>
> eps：根的精度；
>
> root：求得的根。

重根迭代法的 MATLAB 代码如下所示：

```
function root=MultiRoot(f,a,b,eps)
%方程表达式：f
%区间左端点：a
%区间右端点：b
%根的精度：eps
%求得的根：root
if(nargin==3)
    eps=1.0e-4;
end
f1=subs(sym(f),findsym(sym(f)),a);
f2=subs(sym(f),findsym(sym(f)),b);
if(f1==0)
    root=a;
end
```

```
if(f2==0)
    root=b;
end
if(f1*f2>0)
    disp('两端点函数值乘积大于 0!');
    return;
else
    tol=1;
    fun=diff(sym(f));                                   %一阶导数
    ddf=diff(sym(fun));                                 %二阶导数
    fa=subs(sym(f),findsym(sym(f)),a);
    fb=subs(sym(f),findsym(sym(f)),b);
    dfa=subs(sym(fun),findsym(sym(fun)),a);
    dfb=subs(sym(fun),findsym(sym(fun)),b);
    if(dfa>dfb)
        root=a;
    else
        root=b;
    end
    while(tol>eps)
        r1=root;
        fx=subs(sym(f),findsym(sym(f)),r1);
        dfx=subs(sym(fun),findsym(sym(fun)),r1);        %一阶导数值
        ddfx=subs(sym(ddf),findsym(sym(ddf)),r1);       %二阶导数值
        root=r1-fx*dfx/(dfx*dfx-fx*ddfx);               %迭代公式
        tol=abs(root-r1);
    end
end
```

例 9-25 重根迭代法应用实例。用重根迭代法、两步迭代法和弦截法求方程 $x^2(\sin x-x+2)=0$ 在区间[-2,3]上的一个根。

解：在 MATLAB 命令窗口中输入：

```
>> r=MultiRoot('(sin(x)-x+2)*x*x',-2,3,1.0e-8)
r = 0
>> r=TwoStep('(sin(x)-x+2)*x*x',-2,3,1.0e-8)
r = -1.0147e-005
>> r=Secant('(sin(x)-x+2)*x*x',-2,3,1.0e-8)
r =    2.5542
```

方程 $x^2(\sin x-x+2)=0$ 有一个二重根 0，用其他方法得出的精度都不好，而用特殊的重根迭代法可以得到精确的结果。

9.3 小结

本章介绍的方法有的适用于代数多项式方程，有的适用于所有的方程，不过不管方程的形式怎样，在求解方程之前，预先估计根的范围非常重要，这不仅有助于选取正确的方法，而且还能节省求解时间。

第 10 章　非线性方程组求解

非线性方程组的数值解法大多来源于非线性方程的解法，只不过数的计算换成了矩阵的计算。非线性方程组的数值解法中涉及一个重要的矩阵——非线性方程组的雅可比矩阵，它在很大程度上决定了数值求解算法的收敛性和求解精度。

通过本章，读者不仅能掌握常见的非线性方程组的求解算法，而且还能熟练使用 MATLAB 编程来实现这些算法。

10.1 不动点迭代法

非线性方程组的不动点迭代法与非线性方程的不动点迭代法基本一样。

$$\boldsymbol{x}_n = \boldsymbol{F}(\boldsymbol{x}_{n-1}) + \boldsymbol{x}_{n-1}$$

此处公式中的变量皆为向量。

不动点迭代法的 MATLAB 代码介绍如下。

> 在 MATLAB 中编程实现的不动点迭代法的函数为：mulStablePoint
> 功能：用不动点迭代法求非线性方程组的一组解
> 调用格式：[r,n]=mulStablePoint(F,x0,eps)
> 其中，F：非线性方程组；
> 　　x0：初始解；
> 　　eps：解的精度；
> 　　r：求得的一组解；
> 　　n：迭代步数。

不动点迭代法的 MATLAB 程序代码如下所示：

```
function [r,n]=mulStablePoint(F,x0,eps)
%非线性方程组：f
%初始解：a
%解的精度：eps
%求得的一组解：r
%迭代步数：n
if nargin==2
   eps=1.0e-6;
end
n=1;
tol=1;
```

```
while tol>eps
    r= subs(F,findsym(F),x0);                                    %迭代公式
    tol=norm(r-x0);                 %注意矩阵的误差求法，norm 为矩阵的欧几里德范数
    n=n+1;
    x0=r;
    if(n>100000)                    %迭代步数控制
        disp('迭代步数太多，可能不收敛！');
        return;
    end
end
```

例 10-1 不动点迭代法解非线性方程组应用实例。用不动点迭代法求下面方程组的一组解。

$$\begin{cases}0.5\sin x_1 + 0.1\cos(x_1 x_2) - x_1 = 0\\ 0.5\cos x_1 - 0.1\cos x_2 - x_2 = 0\end{cases}$$

初始迭代值取(0,0)。

解：在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = [0.5*sin(x)+0.1*cos(x*y)-x;0.5*cos(x)-0.1*cos(y)-y];
>> [r,n]=mulStablePoint(z,[0 0])
r = 0.1981
    0.3980
n =  19
```

由计算结果可知，初始迭代值取(0,0)时，用 19 步迭代，得到原方程组的一组解(0.1981，0.3980)。

10.2 牛顿法

设非线性方程组为 $F(x_1,x_2,\cdots x_n)=0$，其中 $x=[x_1 \quad x_2 \quad x_3 \quad \cdots \quad x_n]^T$，

$$\mathrm{F}=\begin{bmatrix}f_1(x_1,x_2,\cdots x_n)\\ f_2(x_1,x_2,\cdots x_n)\\ \vdots\\ f_n(x_1,x_2,\cdots x_n)\end{bmatrix}$$

牛顿法的迭代公式为：

$$x^{n+1}=x^n-\left(F^{'}(x^n)\right)^{-1}F(x^n)$$

求解步骤为：

❶ 给出初始值 x^0；

❷ 对 $n=1,2,3,\cdots$，计算 $F(x^n)$ 和 $F^{'}(x^n)$；

❸ 求出 x^{n+1}，并进行精度控制。

更一般的牛顿法迭代公式为：

$$x^{n+1}=x^n-\left(M(x^n)\right)^{-1}F(x^n)\text{，}M\text{ 为任意矩阵}$$

当 $M(x^n)=F'(x^0)$ 时，就得到简化牛顿法。

在 MATLAB 中编程实现的非线性方程组的牛顿法的函数为：mulNewton

功能：用牛顿法求非线性方程组的一组解

调用格式：[r,n]= mulNewton (F,x0,eps)

其中，F：非线性方程组；

x0：初始解；

eps：解的精度；

r：求得的一组解；

n：迭代步数。

牛顿法的 MATLAB 代码如下所示：

```
    function [r,n]=mulNewton(F,x0,eps)
    %非线性方程组：f
    %初始解：a
    %解的精度：eps
    %求得的一组解：r
    %迭代步数：n
if nargin==2
   eps=1.0e-4;
end
x0 = transpose(x0);
Fx = subs(F,findsym(F),x0);
dF = Jacobian(F);
dFx = subs(dF,findsym(dF),x0);
r=x0-Fx/dFx;
n=1;
tol=1;
while tol>eps
   x0=r;
   Fx = subs(F,findsym(F),x0);
   dFx = subs(dF,findsym(dF),x0);
   r=x0-Fx/dFx;                          %核心迭代公式
   tol=norm(r-x0);
   n=n+1;
   if(n>100000)                          %迭代步数控制
      disp('迭代步数太多，可能不收敛！');
      return;
   end
end
```

在 MATLAB 中编程实现的非线性方程组的简化牛顿法的函数为：mulSimNewton

功能：用简化牛顿法求非线性方程组的一组解

调用格式：[r,n]= mulSimNewton (F,x0,eps)

其中，F：非线性方程组；

x0：初始解；

eps：解的精度；

r：求得的一组解；

n：迭代步数。

简化牛顿法的 MATLAB 代码如下所示：

```
function [r,n]=mulSimNewton(F,x0,eps)
%非线性方程组：f
%初始解：a
%解的精度：eps
%求得的一组解：r
%迭代步数：n
if nargin==2
   eps=1.0e-6;
end
x0 = transpose(x0);
Fx = subs(F,findsym(F),x0);
dF = Jacobian(F);
c = subs(dF,findsym(dF),x0);
r=x0-inv(c)*Fx;
n=1;
tol=1;
while tol>eps
   x0=r;
   Fx = subs(F,findsym(F),x0);
   r=x0-inv(c)*Fx;                          %核心迭代公式
   tol=norm(r-x0);
   n=n+1;
   if(n>100000)                             %迭代步数控制
      disp('迭代步数太多，可能不收敛！');
      return;
   end
end
```

例 10-2 牛顿法解非线性方程组应用实例。用牛顿法和简化牛顿法求下面方程组的一组解。

$$\begin{cases} 0.5\sin x_1 + 0.1\cos(x_1 x_2) - x_1 = 0 \\ 0.5\cos x_1 - 0.1\cos x_2 - x_2 = 0 \end{cases}$$

初始迭代值取(0,0)。

解： 在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = [0.5*sin(x)+0.1*cos(x*y)-x;0.5*cos(x)-0.1*cos(y)-y];
>> [r,n]=mulNewton(z,[0 0])
```

```
r =  0.1981
    0.3980
n =  3
  >> [r,n]=mulSimNewton(z,[0 0])
     r =  0.1981
          0.3980
n =   5
```

由计算结果可知，初始迭代值取(0,0)时，分别用 3 步和 5 步迭代，牛顿法和简化牛顿法就得到方程组的一组解(0.1981,0.3980)。

从上面的例子可以看出，牛顿法的收敛速度很快，收敛步数和初始值的选取有很大的关系，初始值选得好，很少的步骤就可以得到根，选得不好可能会不收敛。

10.3 离散牛顿法

在使用牛顿法时，为了避免计算 $F'(x^n)$，可用差商代替偏导数，这就是离散牛顿法，即

$$F'(x) \approx J(x,h) = \begin{bmatrix} \dfrac{f_1(x+h_1\mathrm{e}_1)-f_1(x)}{h_1} & \cdots & \dfrac{f_1(x+h_n\mathrm{e}_n)-f_1(x)}{h_n} \\ \vdots & \ddots & \vdots \\ \dfrac{f_n(x+h_1\mathrm{e}_1)-f_n(x)}{h_1} & \cdots & \dfrac{f_n(x+h_n\mathrm{e}_n)-f_n(x)}{h_n} \end{bmatrix}$$

其中 e 为单位向量，且 $\mathrm{e}_k(k=1,2,\cdots,n)$ 的离散第 k 个分量为 1，其余分量为 0，牛顿法的迭代公式为：

$$x^{n+1} = x^n - J(x^n,h^n)^{-1}F(x^n)$$

其中 $h=(h_1,h_2,\cdots,h_n)^T$。

当 $h=(f_1(x^n),f_2(x^n),\cdots,f_n(x^n))$ 时，就是牛顿-斯蒂芬森法。

在 MATLAB 中编程实现的非线性方程组的离散牛顿法的函数为：mulDiscNewton

功能：用离散牛顿法求非线性方程组的一组解

调用格式：[r,m]= mulDiscNewton (F,x0,h,eps)

其中，F：非线性方程组；

x0：初始解；

h：数值微分增量步大小；

eps：解的精度；

r：求得的一组解；

n：迭代步数。

离散牛顿法的 MATLAB 代码如下所示：

```
function [r,m]=mulDiscNewton(F,x0,h,eps)
%非线性方程组：f
```

```
%初始解: x0
%数值微分增量步大小: h
%解的精度: eps
%求得的一组解: r
%迭代步数: n
format long;
if nargin==3
    eps=1.0e-8;
end
n = length(x0);
fx = subs(F,findsym(F),x0);
J = zeros(n,n);
for i=1:n
    x1 = x0;
    x1(i) = x1(i)+h(i);
    J(:,i) = (subs(F,findsym(F),x1)-fx)/h(i);
end
r=transpose(x0)-inv(J)*fx;
m=1;
tol=1;
while tol>eps
    xs=r;
    fx = subs(F,findsym(F),xs);
    J = zeros(n,n);
    for i=1:n
        x1 = xs;
        x1(i) = x1(i)+h(i);
        J(:,i) = (subs(F,findsym(F),x1)-fx)/h(i);
    end
    r=xs-inv(J)*fx;                                    %核心迭代公式
    tol=norm(r-xs);
    m=m+1;
    if(m>100000)                                       %迭代步数控制
        disp('迭代步数太多，可能不收敛！');
        return;
    end
end
format short;
```

在 MATLAB 中编程实现的非线性方程组的牛顿-斯蒂芬森法的函数为：mulNewtonStev

功能：用牛顿-斯蒂芬森法求非线性方程组的一组解

调用格式：[r,m]= mulNewtonStev (F,x0,h,eps)

其中，F：非线性方程组；

x0：初始解；

h：数值微分增量步大小；

eps：解的精度；

r：求得的一组解；
n：迭代步数。

牛顿-斯蒂芬森的 MATLAB 代码如下所示：

```
function [r,m]=mulNewtonStev(F,x0,h,eps)
%非线性方程组：f
%初始解：x0
%数值微分增量步大小：h
%解的精度：eps
%求得的一组解：r
%迭代步数：n
format long;
if nargin==3
   eps=1.0e-8;
end
n = length(x0);
fx = subs(F,findsym(F),x0);
J = zeros(n,n);
for i=1:n
   x1 = x0;
   x1(i) = x1(i)+h(i);
   J(:,i) = (subs(F,findsym(F),x1)-fx)/h(i);
end
r=transpose(x0)-inv(J)*fx;
m=1;
tol=1;
while tol>eps
   xs=r;
   fx = subs(F,findsym(F),xs);
   J = zeros(n,n);
   h = fx;
   for i=1:n
      x1 = xs;
      x1(i) = x1(i)+h(i);
      J(:,i) = (subs(F,findsym(F),x1)-fx)/h(i);
   end
   r=xs-inv(J)*fx;                        %核心迭代公式
   tol=norm(r-xs);
   m=m+1;
   if(m>100000)                           %迭代步数控制
      disp('迭代步数太多，可能不收敛！');
      return;
   end
end
format short;
```

例 10-3 离散牛顿法解非线性方程组应用实例。用离散牛顿法和牛顿-斯蒂芬森法求下面方程组的一组解。

$$\begin{cases} x^2+xy-2=0 \\ y^2-3x+2=0 \end{cases}$$

初始迭代值取(0,0)。

解： 在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = [x^2+x*y-2;y^2-3*x+2];
>> r=mulDiscNewton(z,([0 0]),[0.1 0.1])
r =  1.0000
    1.0000
m =    17
>> r=mulNewtonStev(z,([0 0]),[0.1 0.1])
r =   1
      1
m =   597
```

此题中，离散牛顿法比牛顿-斯蒂芬森法快了许多。在实际计算时，h 不宜太小，取0.01 至 0.1 之间即可。

10.4 牛顿-松弛型迭代法

在牛顿法中，每一步迭代都要解如下形式的一个线性方程组：

$$F'(x^k)\Delta x^k = -F(x^k)$$

其中 Δx^k 结为解的校正量，如果采用解线性方程组的各种数值方法求解上式，结合各种解线性方程组的方法可得到牛顿-松弛型迭代法。

10.4.1 牛顿-雅可比迭代法

牛顿-雅可比迭代法的迭代步骤如下：

$$A_k = F'(x^k)$$

$$A_k = D - C$$

$$H = D^{-1}C$$

$$x^{k+1} = x^k - (H^{l-1} + H^{l-2} + \cdots + \mathrm{I})D^{-1}F(x^k)$$

其中 $\mathrm{F}'(\mathrm{x}^k)$ 为非线性方程组的雅可比矩阵，用离散法求得，D 为对角矩阵。

在 MATLAB 中编程实现的非线性方程组的牛顿-雅可比迭代法的函数为：mulMix

功能：用牛顿-雅可比迭代法求非线性方程组的一组解

调用格式：[r,m]= mulMix F,x0,h,l,eps)

其中，F：非线性方程组；

x0：初始解；

h：数值微分增量步大小；

l：雅可比迭代参量；

eps：解的精度；
r：求得的一组解；
n：迭代步数。

牛顿-雅可比迭代法的 MATLAB 代码如下所示：

```
function [r,m]=mulMix(F,x0,h,l,eps)
%非线性方程组：f
%初始解：x0
%数值微分增量步大小：h
%l：雅可比迭代参量
%解的精度：eps
%求得的一组解：r
%迭代步数：n
if nargin==4
    eps=1.0e-4;
end
n = length(x0);
J = zeros(n,n);
Fx = subs(F,findsym(F),x0);
for i=1:n
    x1 = x0;
    x1(i) = x1(i)+h(i);
    J(:,i) = (subs(F,findsym(F),x1)-Fx)/h(i);
end
D = diag(diag(J));
C = D - J;
inD = inv(D);
H = inD*C;
Hm = eye(n,n);
for i=1:l-1
    Hm = Hm + power(H,i);
end
dr = Hm*inD*Fx;
r = transpose(x0)-dr;
m=1;
tol=1;
while tol>eps
    x0=r;
    Fx = subs(F,findsym(F),x0);
    J = zeros(n,n);
    for i=1:n
        x1 = x0;
        x1(i) = x1(i)+h(i);
        J(:,i) = (subs(F,findsym(F),x1)-Fx)/h(i);
    end
    D = diag(diag(J));
    C = D - J;
    inD = inv(D);
    H = inD*C;
```

```
    Hm = eye(n,n);
    for i=1:l-1
        Hm = Hm + power(H,i);
    end
    dr = Hm*inD*Fx;
    r = x0-dr;                                        %核心迭代公式
    tol=norm(r-x0);
    m=m+1;
    if(m>100000)                                      %迭代步数控制
        disp('迭代步数太多，可能不收敛！');
        return;
    end
end
```

例 10-4 牛顿-雅可比迭代法解非线性方程组应用实例。用牛顿-雅可比迭代法求下面方程组的一组解。

$$\begin{cases} x^2+xy-2=0 \\ y^2-3x+2=0 \end{cases}$$

初始迭代值取(2,5)。

解：在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = [x^2+x*y-2;y^2-3*x+2];
>> [r,m]=mulMix(z,([2 5]),([0.1 0.1]),2)
r =   1.0000
     1.0000
m =   12
```

迭代 12 步，牛顿-雅可比迭代法得到了一组解(1,1)。

10.4.2 牛顿-SOR 迭代法

牛顿-SOR 迭代法的迭代步骤如下：

$$A_k = F^{'}(x^k)$$

$$A_k = D - L - U$$

$$H = (D - wL)^{-1}[(1-w)\mathrm{D} + wU]$$

$$x^{k+1} = x^k - w(H^{l-1} + H^{l-2} + \cdots + \mathrm{I})(D - wL)^{-1}F(x^k)$$

其中 $F^{'}(x^k)$ 为非线性方程组的雅可比矩阵，用离散法求得；D 为对角矩阵，L 和 U 分别为上三角阵和下三角阵，w 为迭代参数。

在 MATLAB 中编程实现的牛顿-SOR 迭代法的函数为：mulNewtonSOR

功能：用牛顿-SOR 迭代法求非线性方程组的一组解

调用格式：[r,m]=mulNewtonSOR(F,x0,w,h,eps)

其中，F：方程组；

x0：方程组的初始解；
w：SOR 迭代因子；
h：数值微分参数；
eps：根的精度；
r：求得的一个解；
m：迭代步数。

牛顿-SOR 迭代法的 MATLAB 程序代码如下所示：

```
function [r,m]=mulNewtonSOR(F,x0,w,h,eps)
%方程组：f
%方程组的初始解：x0
% SOR迭代因子：w
%数值微分参数：h
%根的精度：eps
%求得的一组解：root
%迭代步数：m
function [r,m]=mulNewtonSOR(F,x0,w,h,l,eps)
if nargin==5
    eps=1.0e-4;
end
n = length(x0);
J = zeros(n,n);
Fx = subs(F,findsym(F),x0);
for i=1:n
    x1 = x0;
    x1(i) = x1(i)+h(i);
    J(:,i) = (subs(F,findsym(F),x1)-Fx)/h(i);
end
D = diag(diag(J));
L = -tril(J-D);
U = -triu(J-D);
inD = inv(D-w*L);
H = inD*(D - w*D+w*L);;
Hm = eye(n,n);
for i=1:l-1
    Hm = Hm + power(H,i);
end
dr = w*Hm*inD*Fx;
r = transpose(x0)-dr;
m=1;
tol=1;
while tol>eps
    x0=r;
    Fx = subs(F,findsym(F),x0);
    J = zeros(n,n);
    for i=1:n
        x1 = x0;
```

```
        x1(i) = x1(i)+h(i);
        J(:,i) = (subs(F,findsym(F),x1)-Fx)/h(i);
    end
    D = diag(diag(J));
    L = -tril(J-D);
    U = -triu(J-D);
    inD = inv(D-w*L);
    H = inD*(D - w*D+w*L);;
    Hm = eye(n,n);
    for i=1:l-1
        Hm = Hm + power(H,i);
    end
    dr = w*Hm*inD*Fx;
    r = x0-dr;                                  %核心迭代公式
    tol=norm(r-x0);
    m=m+1;
    if(m>100000)                                %迭代步数控制
        disp('迭代步数太多，可能不收敛！');
        return;
    end
end
```

例 10-5 牛顿-SOR 迭代法解非线性方程组应用实例。用牛顿-SOR 迭代法求下面方程组的一组解。

$$\begin{cases} x^2 + xy - 2 = 0 \\ y^2 - 3x + 2 = 0 \end{cases}$$

初始迭代值取(2,5)。

解： 在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = [x^2+x*y-2;y^2-3*x+2];
>> [r,m]=mulNewtonSOR(z,[2 5],1.6,[0.1 0.1],2)
r =  1.0000
     1.0000
m =  49
>> [r,m]=mulNewtonSOR(z,[2 5],1.7,[0.1 0.1],2)
r =  0.9998
     1.0003
m =  29
>> [r,m]=mulNewtonSOR(z,[2 5],1.8,[0.1 0.1],2)
r =  0.9997
     1.0004
m =  40
```

在本例中，如果 w 太小，就会发现收敛速度非常慢，小于 1.5 就会出现这种现象。因此当发现收敛速度很慢时，采取的对策是将 h 取得非常接近 2，或者非常接近 1.

10.5 牛顿下山法

牛顿下山法的迭代公式为：

$$x^{n+1} = x^n - \omega\left(F'(x^n)\right)^{-1} F(x^n)$$

ω 的取值范围为 $0<\omega\leqslant 1$，为了保证收敛，还要求 ω 的取值使得：

$$\left\|F(x^{n+1})\right\| < \left\|F(x^n)\right\|$$

可以用逐次减半法来确定 ω。为了减少计算量，还可以用差商来代替偏导数。

> 在 MATLAB 中编程实现的非线性方程组的牛顿下山法的函数为：mulDNewton
>
> 功能：用牛顿下山法求非线性方程组的一组解
>
> 调用格式：[r,m]= mulDNewton (F,x0,eps)
>
> 其中，F：非线性方程组；
>
> x0：初始解；
>
> eps：解的精度；
>
> r：求得的一组解；
>
> n：迭代步数。

牛顿下山法的 MATLAB 代码如下所示：

```
function [r,m]=mulDNewton(F,x0,eps)
%非线性方程组：F
%初始解：x0
%解的精度：eps
%求得的一组解：r
%迭代步数：n
if nargin==2
   eps=1.0e-4;
end
x0 = transpose(x0);
dF = Jacobian(F);
m=1;
tol=1;
while tol>eps
   ttol=1;
   w=1;
   Fx = subs(F,findsym(F),x0);
   dFx = subs(dF,findsym(dF),x0);
   F1=norm(Fx);
   while ttol>=0                          %下面的循环是选取下山因子 w 的过程
      r=x0-w*inv(dFx)*Fx;                 %核心的迭代公式
      Fr = subs(F,findsym(F),r);
      ttol=norm(Fr)-F1;
      w=w/2;
```

```
    end
    tol=norm(r-x0);
    m=m+1;
    x0=r;
    if(n>100000)                           %迭代步数控制
        disp('迭代步数太多，可能不收敛！');
        return;
    end
end
```

例 10-6 牛顿下山法解非线性方程组应用实例。用牛顿下山法求下面方程组的一组解。

$$\begin{cases} 0.5\sin x_1 + 0.1\cos(x_1 x_2) - x_1 = 0 \\ 0.5\cos x_1 - 0.1\sin x_2 - x_2 = 0 \end{cases}$$

初始迭代值取(0,0)。

解： 在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = [0.5*sin(x)+0.1*cos(x*y)-x;0.5*cos(x)-0.1*sin(y)-y];
>> [r,m]=mulDNewton(z,[0 0])
r =  0.1979
     0.4470
m =  4
```

由计算结果可知，初始迭代值取(0,0)时，用 4 步迭代得到了方程组的一组解(0.1979,0.4470)，牛顿下山法也是一种快速且有效的求解非线性方程组的方法。

10.6 割线法

设 x^m 为方程组的第 m 次迭代求出的解，取 $x^{m,0}=x^m$，令
$\begin{cases} H_m = (x^{m,1}-x^m, x^{m,2}-x^m, \cdots, x^{m,n}-x^m) \\ \Gamma_m = (F(x^{m,1})-F(x^m), F(x^{m,2})-F(x^m), \cdots, F(x^{m,n})-F(x^m)) \end{cases}$，则方程组的一般割线法的公式如下：

$$\mathrm{x}^{m+1} = \mathrm{x}^m - H_m \Gamma_m^{-1} \mathrm{F}(\mathrm{x}^m)$$

如果取 $H_m = diag((x_1^{m-1}-x_1^m),(x_2^{m-1}-x_2^m),\cdots,(x_n^{m-1}-x_n^m))$，则得到两点割线法公式为：

$$\mathrm{x}^{m+1} = \mathrm{x}^m - A^{-1}\mathrm{F}(\mathrm{x}^m)$$

其中

$$A = \left(\frac{1}{x_1^{m-1}-x_1^m}[\mathrm{F}(\mathrm{x}^m + (x_1^{m-1}-x_1^m)\mathrm{e}_1) - \mathrm{F}(\mathrm{x}^m)], \cdots, \frac{1}{x_n^{m-1}-x_n^m}[\mathrm{F}(\mathrm{x}^m + (x_n^{k-1}-x_n^k)\mathrm{e}_n) - \mathrm{F}(\mathrm{x}^m)]\right.$$

两点割线法需要给定两个初始解的近似值。

若取

$$H_m = \begin{bmatrix} h_1^m & h_1^m & \cdots & h_1^m \\ & h_2^m & \cdots & h_2^m \\ & & \ddots & \vdots \\ 0 & & & h_n^m \end{bmatrix}$$

其中 $h_j^m = x_j^{m-1} - x_j^m (j=1,2,\cdots,n)$，则得到两点割线法的另一种形式：

$$x^{m+1} = x^m - A^{-1}F(x^m)$$

其中

$$A = (\frac{1}{h_1^m}[F(x^m + h_1^m \mathrm{e}_1) - F(x^m)],\cdots,$$

$$\frac{1}{h_n^m}[F(x^m + \sum_{i=1}^{n} h_i^m \mathrm{e}_i) - F(x^m + \sum_{i=1}^{n-1} h_i^m \mathrm{e}_i)])$$

在 MATLAB 中编程实现的非线性方程组的两点割线法的函数为：mulGXF1

功能：用两点割线法的第一种形式求非线性方程组的一组解

调用格式：[r,m]=mulGXF1(F,x0,x1,eps)

其中，F：非线性方程组；

x0：初始解；

x1：初始解；

eps：解的精度；

r：求得的一组解；

m：迭代步数。

两点割线法第一种形式的 MATLAB 代码如下所示：

```
    function [r,m]=mulGXF1(F,x0,x1,eps)
    %非线性方程组：F
    %初始解：x0
    %初始解：x1
    %解的精度：eps
    %求得的一组解：r
    %迭代步数：m
format long;
if nargin==3
    eps=1.0e-4;
end
x0 = transpose(x0);
x1 = transpose(x1);
n = length(x0);
fx = subs(F,findsym(F),x0);
fx1 = subs(F,findsym(F),x1);
h = x0 - x1;
J = zeros(n,n);
```

```
for i=1:n
    xt = x1;
    xt(i) = x0(i);
    J(:,i) = (subs(F,findsym(F),xt)-fx1)/h(i);
end
r=x1-inv(J)*fx1;
m=1;
tol=1;
while tol>eps
    x0 = x1;
    x1 = r;
    fx = subs(F,findsym(F),x0);
    fx1 = subs(F,findsym(F),x1);
    h = x0 - x1;
    J = zeros(n,n);
    for i=1:n
        xt = x1;
        xt(i) = x0(i);
        J(:,i) = (subs(F,findsym(F),xt)-fx1)/h(i);
    end
    r=x1-inv(J)*fx1;
    tol=norm(r-x1);
    m=m+1;
    if(m>100000)                                        %迭代步数控制
        disp('迭代步数太多，可能不收敛！');
        return;
    end
end
format short;
```

在 MATLAB 中编程实现的非线性方程组的两点割线法的函数为：mulGXF2

功能：用两点割线法的第二种形式求非线性方程组的一组解

调用格式：[r,m]=mulGXF2(F,x0,x1,eps)

其中，F：非线性方程组；

x0：初始解；

x1：初始解；

eps：解的精度；

r：求得的一组解；

m：迭代步数。

两点割线法第二种形式的 MATLAB 代码如下所示：

```
function [r,m]=mulGXF2(F,x0,x1,eps)
%非线性方程组：F
%初始解：x0
%初始解：x1
%解的精度：eps
%求得的一组解：r
```

```
%迭代步数: m
format long;
if nargin==3
    eps=1.0e-4;
end
x0 = transpose(x0);
x1 = transpose(x1);
n = length(x0);
fx = subs(F,findsym(F),x0);
fx1 = subs(F,findsym(F),x1);
h = x0 - x1;
J = zeros(n,n);
xt = x1;
xt(1) = x0(1);
J(:,1) = (subs(F,findsym(F),xt)-subs(F,findsym(F),x1))/h(1);
for i=2:n
    xt = x1;
    xt(1:i) = x0(1:i);
    xt_m = x1;
    xt_m(1:i-1) = x0(1:i-1);
    J(:,i) = (subs(F,findsym(F),xt)-subs(F,findsym(F),xt_m))/h(i);
end
r=x1-inv(J)*fx1;
m=1;
tol=1;
while tol>eps
    x0 = x1;
    x1 = r;
    fx = subs(F,findsym(F),x0);
    fx1 = subs(F,findsym(F),x1);
    h = x0 - x1;
    J = zeros(n,n);
    xt = x1;
    xt(1) = x0(1);
    J(:,1) = (subs(F,findsym(F),xt)-subs(F,findsym(F),x1))/h(1);
    for i=2:n
        xt = x1;
        xt(1:i) = x0(1:i);
        xt_m = x1;.
        xt_m(1:i-1) = x0(1:i-1);
        J(:,i) = (subs(F,findsym(F),xt)-subs(F,findsym(F),xt_m))/h(i);
    end
    r=x1-inv(J)*fx1;
    tol=norm(r-x1);
    m=m+1;
    if(m>100000)                                    %迭代步数控制
        disp('迭代步数太多，可能不收敛！');
        return;
    end
end
format short;
```

例 10-7 割线法解非线性方程组应用实例。用两点割线法求下面方程组的一组解。

$$\begin{cases} x^2+y^2-5=0 \\ 2x-y-3=0 \end{cases}$$

解：在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = [x^2+y^2-5;2*x-y-3];
>> [r,m] = mulGXF1(z,[0 1],[4 3])
r =  2.0000
    1.0000
m =  7
>> [r,m] = mulGXF2(z,[0 1],[4 3])
r =  2.0000
    1.0000
m =  7
>> [r,m] = mulGXF2(z,[-3 1],[4 3])
r =  0.4000
   -2.2000
m =  11
```

通过不同的初始解，用两点割线法求出了方程组的两组解：(2,1)和(0.4,-2.2).

10.7 拟牛顿法

拟牛顿法是为了减少计算导数而带来的计算量的一种迭代法，它的迭代过程如下：

$$x^{k+1}=x^k-A_k^{-1}F(x^k);$$

$$y^k=x^{k+1}-x^k;$$

$$z^k=F(x^{k+1})-F(x^k);$$

$$A_{k+1}=A_k+\frac{(z^k-A_k y^k)(y^k)^T}{\left\|y^k\right\|}$$

拟牛顿法的思想是用比较简单的矩阵 A_k 来近似 $F^{'}(x^k)$，A_0 可选为单位阵。

在 MATLAB 中编程实现的拟牛顿法的函数为：mulVNewton

功能：用拟牛顿法求非线性方程组的一组解

调用格式：[r,m]=mulVNewton(F,x0,A,eps)

其中，F：方程组；

x0：方程组的初始解；

A：初始 A 矩阵；

eps：解的精度；

r：求得的一组解；

m：迭代步数。

拟牛顿法的 MATLAB 代码如下所示：

```
function [r,m]=mulVNewton(F,x0,A,eps)
%方程组: F
%方程组的初始解: x0
% 初始矩阵: A
%解的精度: eps
%求得的一组解: r
%迭代步数: m
if nargin==2
    A=eye(length(x0));                          %A 取为单位阵
else
   if nargin==3
        eps=1.0e-4;
    end
end
x0 = transpose(x0);
Fx = subs(F, findsym(F),x0);
r=x0-Fx/A;
m=1;
tol=1;
while tol>eps
    x0=r;
    Fx = subs(F, findsym(F),x0);
    r=x0-Fx/A;
    y=r-x0;
    Fr = subs(F, findsym(F),r);
    z= Fr-Fx;
    A1=A+(z-y*A)'*y/norm(y);                    %调整 A
    A=A1;
    m=m+1;
    if(n>100000)                                %迭代步数控制
        disp('迭代步数太多,可能不收敛! ');
        return;
    end
    tol=norm(r-x0);
end
```

例 10-8 拟牛顿法解非线性方程组应用实例。用拟牛顿法求下面方程组的一组解。

$$\begin{cases}0.5\sin x_1 + 0.1\cos(x_1 x_2) - x_1 = 0\\0.5\cos x_1 - 0.1\sin x_2 - x_2 = 0\end{cases}$$

其初始迭代值取(0.5,0.5)。

解:在 MATLAB 命令窗口中输入:

```
>> syms x y;
>> z = [0.5*sin(x)+0.1*cos(x*y)-x;0.5*cos(x)-0.1*sin(y)-y];
>> [r,m]=mulVNewton([0.5,0.5])
```

输出计算结果为:

```
r =  0.1979
     0.4470
```

```
n =  16
```

由计算结果可知，初始迭代值取(0.5,0.5)时，用 16 步迭代得到了方程组的一组解(0.1980,0.4470)。

拟牛顿法中重要的是要选好 A_0，否则，迭代的效果会很差。

10.8 对称秩 1 算法

对称秩 1 算法本质上也是一种拟牛顿法，其迭代公式和拟牛顿法的迭代公式十分相似：

$$x^{n+1} = x^n - A_n^{-1}F(x^n);$$

$$y^n = x^{n+1} - x^n;$$

$$z^n = F(x^{n+1}) - F(x^n);$$

$$A_{n+1} = A_n + \frac{(z^n - A_n y^n)(z^n - A_n y^n)^T}{(z^n - A_n y^n)^T y^n}$$

在 MATLAB 中编程实现的对称秩 1 算法的函数为：mulRank1

功能：用对称秩 1 算法求非线性方程组的一组解

调用格式：[r,n]=mulRank1(F,x0,A,eps)

其中，F：方程组；

x0：方程组的初始解；

A：初始 A 矩阵；

eps：根的精度；

r：求得的一组解；

n：迭代步数。

对称秩 1 算法的 MATLAB 程序代码如下所示：

```
function [r,n]=mulRank1(F,x0,A,eps)
%方程组：f
%方程组的初始解：x0
%初始 A 矩阵：A
%根的精度：eps
%求得的一组解：r
%迭代步数：n
if nargin==2
    l = length(x0);
    A=eye(l);                                    %A 取为单位阵
    eps=1.0e-4;
else
   if nargin==3
        eps=1.0e-4;
    end
end
fx = subs(F,findsym(F),x0);
```

```
r=transpose(x0)-inv(A)*fx;
n=1;
tol=1;
while tol>eps
    x0=r;
    fx = subs(F,findsym(F),x0);
    r=x0-inv(A)*fx;
    y=r-x0;
    fr = subs(F,findsym(F),r);
    z = fr-fx;
    A1=A+ fr *transpose(fr)/(transpose(fr)*y);                    %调整 A
    A=A1;
    n=n+1;
    if(n>100000)                                         %迭代步数控制
        disp('迭代步数太多，可能不收敛！');
        return;
    end
    tol=norm(r-x0);
end
```

例 10-9 对称秩 1 法解非线性方程组应用实例。用对称秩 1 算法求下面方程组的一组解。

$$\begin{cases} x^2 + xy - 2 = 0 \\ y^2 - 3x + 2 = 0 \end{cases}$$

初始迭代值取(2,5)。

解： 在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = [x^2+x*y-2;y^2-3*x+2];
>> [r,n]=mulRank1(z,[2 5])
r =  1.0000
     1.0000
n =  193
```

用对称秩 1 算法迭代到 193 步才求得一组解(1,1)。

10.9 D-F-P 算法

D-F-P 算法是一种秩 2 算法，它的特点是迭代公式中不需要求矩阵的逆，其迭代公式如下：

$$x^{n+1} = x^n - B_n F(x^n);$$

$$y^n = x^{n+1} - x^n;$$

$$z^n = F(x^{n+1}) - F(x^n);$$

$$B_{n+1} = B_n + \frac{y^n (y^n)^T}{(y^n)^T z^n} - \frac{B_n z^n (z^n)^T B_n}{(z^n)^T B_n z^n}$$

在 MATLAB 中编程实现的 D-F-P 算法的函数为：mulDFP

功能：用 D-F-P 算法求非线性方程组的一组解

调用格式：[r,n]= mulDFP (F,x0,B,eps)

其中，F：方程组；

x0：方程组的初始解；

B：初始 B 矩阵；

eps：解的精度；

r：求得的一组解；

n：迭代步数。

D-F-P 算法的 MATLAB 程序代码如下所示：

```
function [r,n]= mulDFP(F,x0,B,eps)
%方程组：F
%方程组的初始解：x0
%初始 B 矩阵：B
%根的精度：eps
%求得的一组解：r
%迭代步数：n
if nargin==2
    l = length(x0);
    B=eye(l);                          %A 取为单位阵
    eps=1.0e-4;
else
   if nargin==3
        eps=1.0e-4;
    end
end
fx`= subs(F,findsym(F),x0);
r=transpose(x0)-B*fx;
n=1;
tol=1;
while tol>eps
    x0=r;
    fx = subs(F,findsym(F),x0);
    r=x0-B*fx;
    y=r-x0;
    fr = subs(F,findsym(F),r);
    z = fr-fx;
    B1=B+ y*y'/(y'*z)-B*z*z'*B/(z'*B*z);          %调整 A
    B=B1;
    n=n+1;
    if(n>100000)                             %迭代步数控制
        disp('迭代步数太多，可能不收敛！');
        return;
    end
    tol=norm(r-x0);
end
```

例 10-10 D-F-P 法解非线性方程组应用实例。用 D-F-P 算法求下面方程组的一组解。

$$\begin{cases} x^2 + xy - 2 = 0 \\ y^2 - 3x + 2 = 0 \end{cases}$$

初始迭代值取(0,5)。

解： 在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = [x^2+x*y-2;y^2-3*x+2];
>> [r,n]=mulDFP(z,[0 5])
r =  1.0000
     1.0000
n =  53
```

与对称秩 1 算法相比，D-F-P 算法快了许多。

10.10 B-F-S 算法

B-F-S 算法也是一种秩 2 算法，它的特点是迭代公式中也不需要求矩阵的逆，其迭代公式如下：

$$x^{n+1} = x^n - B_n F(x^n);$$
$$y^n = x^{n+1} - x^n;$$
$$z^n = F(x^{n+1}) - F(x^n);$$
$$\mu_n = 1 + \frac{(z^n)^T B_n z^n}{(y^n)^T z^n}$$
$$B_{n+1} = B_n + \frac{\mu_n y^n (y^n)^T - B_n z^n (y^n)^T - y^n (z^n)^T B_n}{(y^n)^T z^n}$$

在 MATLAB 中编程实现的 B-F-S 算法的函数为：mulBFS

功能：用 B-F-S 算法求非线性方程组的一组解

调用格式：[r,n]= mulBFS (F,x0,A,eps)

其中，F：方程组；

x0：方程组的初始解；

B：初始 B 矩阵；

eps：解的精度；

r：求得的一组解；

n：迭代步数。

B-F-S 算法的 MATLAB 程序代码如下所示：

```
function [r,n]= mulBFS (F,x0,B,eps)
%方程组：F
```

```
%方程组的初始解: x0
%初始 A 矩阵: B
%根的精度: eps
%求得的一个解: r
%迭代步数: n
if nargin==2
    l = length(x0);
    B=eye(l);                                    %A 取为单位阵
    eps=1.0e-4;
else
   if nargin==3
        eps=1.0e-4;
    end
end
fx = subs(F,findsym(F),x0);
r=transpose(x0)-B*fx;
n=1;
tol=1;
while tol>eps
    x0=r;
    fx = subs(F,findsym(F),x0);
    r=x0-B*fx;
    y=r-x0;
    fr = subs(F,findsym(F),r);
    z = fr-fx;
    u = 1 + z'*B*z/(y'*z);
    B1= B+ (u*y*y'-B*z*y'-y*z'*B)/(y'*z);               %调整 A
    B=B1;
    n=n+1;
    if(n>100000)                                 %迭代步数控制
        disp('迭代步数太多, 可能不收敛! ');
        return;
    end
    tol=norm(r-x0);
end
```

例 10-11 B-F-S 法解非线性方程组应用实例。用 B-F-S 算法求下面方程组的一组解。

$$\begin{cases} x^2 + xy - 2 = 0 \\ y^2 - 3x + 2 = 0 \end{cases}$$

初始迭代值取(−3,2)。

解: 在 MATLAB 命令窗口中输入:

```
>> syms x y;
>> z = [x^2+x*y-2;y^2-3*x+2];
>> [r,n]=mulBFS(z,[-3 2])
r =  1.0000
     1.0000
n =  309
```

用 B-F-S 算法迭代 309 步，得到了一组解(1,1)。

10.11 数值延拓法

数值延拓法是延拓法的一种，它的一般迭代公式如下：

$$\begin{cases} x^{i,j+1} = x^{i,j} - [H_x(x^{i,j},t_i)]^{-1} H(x^{i,j},t_i) \\ x^{1,0} = x^0, x^{i+1,0} = x^{i,m_i} \\ x^{k+1} = x^k - [H_x(x^k,1)]^{-1} H(x^k,1) \\ x^N = x^{N,0} \end{cases} \quad \begin{matrix} j = 0,1,\cdots,m_j - 1 \\ i = 1,2,\cdots,N-1 \\ k = N, N+1,\cdots \end{matrix}$$

当取 $m_j = 1, t_i = \dfrac{i}{N}$ 时得到如下的具体表达式：

$$\begin{cases} x^{k+1} = x^k - \left(F^{'}(x^k)\right)^{-1} [F(x^k) - (1 - \dfrac{k}{N})F(x^0)], & (k = 0,1,\cdots,N-1) \\ x^{k+1} = x^k - \left(F^{'}(x^k)\right)^{-1} F(x^k), & (k = N, N+1,\cdots) \end{cases}$$

在 MATLAB 中编程实现的数值延拓法的函数为：mulNumYT

功能：用数值延拓法求非线性方程组的一组解

调用格式：[r,m]=mulNumYT(F,x0,h,N,eps)

其中，F：方程组；

x0：方程组的初始解；

h：数值微分增量步；

N：迭代公式中的 N；

eps：解的精度；

r：求得的一组解；

m：迭代步数。

数值延拓法的 MATLAB 程序代码如下所示：

```
function [r,m]=mulNumYT(F,x0,h,N,eps)
%方程组：F
%方程组的初始解：x0
%数值微分增量步：h
%迭代公式中的N：N
%解的精度：eps
%求得的一组解：r
%迭代步数：m
format long;
if nargin==4
    eps=1.0e-8;
end
n = length(x0);
fx0 = subs(F,findsym(F),x0);
```

```
x0 = transpose(x0);
J = zeros(n,n);
for k=0:N-1
    fx = subs(F,findsym(F),x0);
    for i=1:n
        x1 = x0;
        x1(i) = x1(i)+h(i);
        J(:,i) = (subs(F,findsym(F),x1)-fx)/h(i);
    end
    inJ = inv(J);
    r=x0-inJ*(fx-(1-k/N)*fx0);
    x0 = r;
end
m=1;
tol=1;
while tol>eps
    xs=r;
    fx = subs(F,findsym(F),xs);
    J = zeros(n,n);
    for i=1:n
        x1 = xs;
        x1(i) = x1(i)+h(i);
        J(:,i) = (subs(F,findsym(F),x1)-fx)/h(i);
    end
    r=xs-inv(J)*fx;                          %核心迭代公式
    tol=norm(r-xs);
    m=m+1;
    if(m>100000)                             %迭代步数控制
        disp('迭代步数太多，可能不收敛！');
        return;
    end
end
format short;
```

例 10-12 数值延拓法解非线性方程组应用实例。用数值延拓法求下面方程组的一组解。

$$\begin{cases} x^2+y^2-5=0 \\ 2x-y-3=0 \end{cases}$$

初始迭代值取(−3,5)。

解：在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = [x^2+y^2-5;2*x-y-3];
>> [r,m] = mulNumYT(z,[-3 5],[0.1 0.1],10)
r =  2.0000
    1.0000
m =  9
```

此题用数值延拓法经过 9 步迭代就得到一组解(2,1)。

10.12 参数微分法

参数微分法是把非线性方程组转化为一个如下形式的微分方程的初值问题：

$$\begin{cases} x'(t) = -[H_x(x,t)]^{-1} H_t(x,t) \\ x(0) = x^0 \end{cases}$$

且其中 $H(x,t)$ 是 x,t 的函数，且有：

$$\begin{cases} H(x,0) = F_0(x) \\ H(x,1) = F(x) \end{cases}$$

如果取 $H(x,t) = F(x) - (1-t)F(x^0), t \in (0,1)$，则得到

$$\begin{cases} x'(t) = -[F'(x)]^{-1} F(x^0) \\ x(0) = x^0 \end{cases}$$

然后就可以用求解微分方程初值问题的各种方法求解上面的微分方程。常用的方法有欧拉法和中点求积法。

- 欧拉法

$$x^{k+1} = x^k - \frac{1}{N} F'(x^k)^{-1} F(x^0), \quad k = 0,1,2,\cdots,N-1$$

- 中点求积法

$$\begin{cases} x^1 = x^0 - \frac{1}{N} F'(x^k)^{-1} F(x^0) \\ y^k = x^k + \frac{1}{2}(x^k - x^{k-1}) \quad , \quad k = 1,2,\cdots,N-1 \\ x^{k+1} = x^k - \frac{1}{N} F'(y^k)^{-1} F(x^0) \end{cases}$$

在 MATLAB 中编程实现的欧拉法的函数为：DiffParam1

功能：用参数微分法中的欧拉法求非线性方程组的一组解

调用格式：r=DiffParam1(F,x0,h,N)

其中，F：方程组；

x0：方程组的初始解；

h：数值微分增量步；

N：迭代步数；

r：求得的一组解。

欧拉法的 MATLAB 程序代码如下所示：

```
function r=DiffParam1(F,x0,h,N)
%非线性方程组：F
%初始解：x0
%数值微分增量步大小：h
```

```
%迭代步数：N
%求得的一组解：r
x0 = transpose(x0);
n = length(x0);
ht = 1/N;
Fx0  = subs(F,findsym(F),x0);
for k=1:N
    Fx  = subs(F,findsym(F),x0);
    J = zeros(n,n);
    for i=1:n
        x1 = x0;
        x1(i) = x1(i)+h(i);
        J(:,i) = (subs(F,findsym(F),x1)-Fx)/h(i);
    end
    inJ = inv(J);
    r = x0 - ht*inJ*Fx0;
    x0 = r;
end
```

在 MATLAB 中编程实现的中点积分法的函数为：DiffParam2

功能：用参数微分法中的中点积分法求非线性方程组的一组解

调用格式：r=DiffParam2(F,x0,h,N)

其中，F：方程组；

x0：方程组的初始解；

h：数值微分增量步；

N：迭代步数；

r：求得的一组解。

中点积分法的 MATLAB 程序代码如下所示：

```
function r=DiffParam2(F,x0,h,N)
%非线性方程组：F
%初始解：x0
%数值微分增量步大小：h
%迭代步数：N
%求得的一组解：r
x0 = transpose(x0);
n = length(x0);
ht = 1/N;
Fx0  = subs(F,findsym(F),x0);
J = zeros(n,n);
for i=1:n
    xt = x0;
    xt(i) = xt(i)+h(i);
    J(:,i) = (subs(F,findsym(F),xt)-Fx0)/h(i);
end
inJ = inv(J);
x1 = x0 - ht*inJ*Fx0;
```

```
for k=1:N
    x2 = x1 + (x1-x0)/2;
    Fx2  = subs(F,findsym(F),x2);
    J = zeros(n,n);
    for i=1:n
        xt = x2;
        xt(i) = xt(i)+h(i);
        J(:,i) = (subs(F,findsym(F),xt)-Fx2)/h(i);
    end
    inJ = inv(J);
    r = x1 - ht*inJ*Fx0;
    x0 = x1;
    x1 = r;
end
```

例 10-13 欧拉法解非线性方程组应用实例。用欧拉法求下面方程组的一组解。

$$\begin{cases} x^2 + y^2 - 5 = 0 \\ 2x - y - 3 = 0 \end{cases}$$

初始迭代值取(0,0)。

解： 在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = [x^2+y^2-5;2*x-y-3];
>> r=DiffParam1(z,[0 0],[0.1 0.1],500)
r =  1.9656
     0.9312
```

用欧拉法得到了一组近似解(1.9656,0.9312)，此题的步长为 0.1，如果要得到更精确的解，可以采取更小的步长和更多的迭代步。

例 10-14 中点积分法解非线性方程组应用实例。用中点积分法求下面方程组的一组解。

$$\begin{cases} x^2 + y^2 - 5 = 0 \\ 2x - y - 3 = 0 \end{cases}$$

初始迭代值取(0,0)。

解： 在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = [x^2+y^2-5;2*x-y-3];
>> r=DiffParam2(z,[0 0],[0.1 0.1],500)
r =  1.9662
     0.9265
```

和准确解(2,1)相比，两种参数微分法求出的解都有一定的误差，这是和所采用的求解初值问题的方法有关的。

10.13 最速下降法

最速下降法的算法如下：

❶ 给定一组初始解 x^0；

❷ 令 $\varphi=\sum_{i=1}^{n} f_i^2(x^0)$，对给定的精度 ε，如果 $\varphi<\varepsilon$，则认为 x^0 为方程组的解，否则计算

$$\begin{cases} \dfrac{\partial F}{\partial x_1}=\dfrac{F(x_1^0+h_1,x_2^0,\cdots,x_n^0)-F(x^0)}{h_1} \\ \dfrac{\partial F}{\partial x_2}=\dfrac{F(x_1^0,x_2^0+h_2,\cdots,x_n^0)-F(x^0)}{h_2} \\ \vdots \\ \dfrac{\partial F}{\partial x_n}=\dfrac{F(x_1^0,x_2^0,\cdots,x_n^0+h_n)-F(x^0)}{h_n} \end{cases}$$

其中 $h_i=hx_i^0$，h 为控制收敛的参数。

❸ 计算

$$\lambda=\frac{F(x_1^0,x_2^0,\cdots,x_n^0)}{\sum_{i=1}^{n}(\dfrac{\partial F}{\partial x_i})^2}$$

$$x^1=x^0-(\frac{\partial F}{\partial x_1},\frac{\partial F}{\partial x_2},\cdots,\frac{\partial F}{\partial x_n})*\lambda$$

然后再转❷，直至求出满足精度的解。

在 MATLAB 中编程实现的最速下降法的函数为：mulFastDown

功能：用最速下降法求非线性方程组的一组解

调用格式：[r,m]= mulFastDown (F,x0,h,eps)

其中，F：方程组；

x0：方程组的初始解；

h：数值微分增量步；

eps：解的精度；

r：求得的一组解；

m：迭代步数。

最速下降法的 MATLAB 程序代码如下所示：

```
function [r,m]=mulFastDown(F,x0,h,eps)
%方程组：F
%方程组的初始解：x0
%数值微分增量步：h
```

```
%解的精度：eps
%求得的一组解：r
%迭代步数：m
format long;
if nargin==3
   eps=1.0e-8;
end
n = length(x0);
x0 = transpose(x0);
m=1;
tol=1;
while tol>eps
    fx = subs(F,findsym(F),x0);
    J = zeros(n,n);
    for i=1:n
        x1 = x0;
        x1(i) = x1(i)+h;
        J(:,i) = (subs(F,findsym(F),x1)-fx)/h;
    end
    lamda = fx/sum(diag(transpose(J)*J));
    r=x0-J*lamda;  %核心迭代公式
    fr = subs(F,findsym(F),r);
    tol=dot(fr,fr);
    x0 = r;
    m=m+1;
    if(m>100000)                                         %迭代步数控制
        disp('迭代步数太多，可能不收敛！');
        return;
    end
end
format short;
```

例 10-15 最速下降法解非线性方程组应用实例。用最速下降法求下面方程组的一组解。

$$\begin{cases} 0.5\sin x_1 + 0.1\cos(x_1 x_2) - x_1 = 0 \\ 0.5\cos x_1 - 0.1\sin x_2 - x_2 = 0 \end{cases}$$

初始迭代值取(0,0)。

解： 在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = [0.5*sin(x)+0.1*cos(x*y)-x;0.5*cos(x)-0.1*sin(y)-y];
>> [r,m] = mulFastDown(z,[-3 5],1.0e-6)
r =  0.1977
    0.4470
m =  48
```

最速下降法对于各种方程组的平均效率是很不错的，此题方程组比较复杂，经过 48 步迭代最速下降法得到了一组解(0.1977,0.4470)。

10.14 高斯牛顿法

高斯牛顿法的迭代公式如下：

$$x^{k+1} = x^k - [\nabla F(x^k)^T \nabla F(x^k)]^{-1} \nabla F(x^k)^T F(x^k)$$

其中 $\nabla F(x^k)$ 表示 $F(x)$ 在 x^k 处的梯度。

在 MATLAB 中编程实现的高斯牛顿法的函数为：mulGSND

功能：用高斯牛顿法求非线性方程组的一组解

调用格式：[r,m]= mulGSND (F,x0,h,eps)

其中，F：方程组；

x0：方程组的初始解；

h：数值微分增量步；

eps：解的精度；

r：求得的一组解；

m：迭代步数。

高斯牛顿法的 MATLAB 程序代码如下所示：

```
function [r,m]=mulGSND(F,x0,h,eps)
%方程组：F
%方程组的初始解：x0
%数值微分增量步：h
%解的精度：eps
%求得的一组解：r
%迭代步数：m
format long;
if nargin==3
    eps=1.0e-8;
end
n = length(x0);
x0 = transpose(x0);
m=1;
tol=1;
while tol>eps
    fx = subs(F,findsym(F),x0);
    J = zeros(n,n);
    for i=1:n
        x1 = x0;
        x1(i) = x1(i)+h;
        J(:,i) = (subs(F,findsym(F),x1)-fx)/h;
    end
    DF = inv(transpose(J)*J)*transpose(J);
    r=x0-DF*fx;  %核心迭代公式
    tol=norm(r-x0);
```

```
        x0 = r;
        m=m+1;
        if(m>100000)                                        %迭代步数控制
            disp('迭代步数太多，可能不收敛！');
            return;
        end
    end
    format short;
```

例 10-16 高斯牛顿法解非线性方程组应用实例。用高斯牛顿法求下面方程组的一组解。

$$\begin{cases} x^2 + y - 6 = 0 \\ y^2 - x - 2 = 0 \end{cases}$$

初始迭代值取(−1,3)。

解：在 MATLAB 命令窗口中输入：

```
>>syms x y;
>> z=[x^2+y-6;y^2-x-2];
>> [r,m] = mulGSND(z,[-1 3],1.0e-1)
r =  2.0000
     2.0000
m =  224
```

用高斯牛顿法经过 224 步迭代得到一组解(2.0000,2.0000)。

10.15 共轭梯度法

共轭梯度法的迭代公式如下：

$$\begin{cases} x^{k+1} = x^k + \lambda_k p_k \\ \lambda_k = \dfrac{F(x^k)}{\sum\limits_{i=1}^{n}(\dfrac{\partial F}{\partial x_i})^2} \\ p_{k+1} = -\nabla F(x^{k+1}) + v_k p_k \\ v_k = \dfrac{\nabla F(x^{k+1})^T \nabla F(x^{k+1})}{\nabla F(x^k)^T \nabla F(x^k)} \end{cases}$$

其中，$p_0 = -\nabla F(x^0)$。

在 MATLAB 中编程实现的共轭梯度法的函数为：mulConj

功能：用共轭梯度法求非线性方程组的一组解

调用格式：[r,m]= mulConj (F,x0,h,eps)

其中，F：方程组；

x0：方程组的初始解；

h：数值微分增量步；
eps：解的精度；
r：求得的一组解；
m：迭代步数。

共轭梯度法的 MATLAB 程序代码如下所示：

```
function [r,m]=mulConj(F,x0,h,eps)
%方程组：F
%方程组的初始解：x0
%数值微分增量步：h
%解的精度：eps
%求得的一组解：r
%迭代步数：m
format long;
if nargin==3
   eps=1.0e-6;
end
n = length(x0);
x0 = transpose(x0);
fx0 = subs(F,findsym(F),x0);
p0 = zeros(n,n);
for i=1:n
   x1 = x0;
   x1(i) = x1(i)*(1+h);
   p0(:,i) = -(subs(F,findsym(F),x1)-fx0)/h;
end
m=1;
tol=1;
while tol>eps
    fx = subs(F,findsym(F),x0);
    J = zeros(n,n);
    for i=1:n
        x1 = x0;
        x1(i) = x1(i)+h;
        J(:,i) = (subs(F,findsym(F),x1)-fx)/h;
    end
    lamda = fx/sum(diag(transpose(J)*J));
    r=x0+p0*lamda;  %核心迭代公式
    fr = subs(F,findsym(F),r);
    Jnext = zeros(n,n);
    for i=1:n
        x1 = r;
        x1(i) = x1(i)+h;
        Jnext(:,i) = (subs(F,findsym(F),x1)-fr)/h;
    end
    abs1 = transpose(Jnext)*Jnext;
    abs2 = transpose(J)*J;
    v = abs1/abs2;
```

```
    if (abs(det(v)) < 1)
        p1 = -Jnext+p0*v;
    else
        p1 = -Jnext;
    end
    tol=norm(r-x0);
    p0 = p1;
    x0 = r;
    m=m+1;
    if(m>100000)                                    %迭代步数控制
        disp('迭代步数太多，可能不收敛！');
        return;
    end
end
format short;
```

例 10-17 共轭梯度法解非线性方程组应用实例。用共轭梯度法求下面方程组的一组解。

$$\begin{cases} x^2+y-6=0 \\ y^2-x-2=0 \end{cases}$$

初始迭代值取(4,4)。

解： 在 MATLAB 命令窗口中输入：

```
>>syms x y;
>> z=[x^2+y-6;y^2-x-2];
>> [r,m] = mulConj(z,[4 4],1.0e-1)
r =  2.0000
     2.0000
m =  43
```

共轭梯度法也是一种效率很高的算法，一般都能经过比较少的步数就可得到解。

10.16 阻尼最小二乘法

阻尼最小二乘法的算法如下：

❶ 给出初始值 x^0，阻尼因子 μ_0，缩放常数 $\upsilon>1$；

❷ 计算 $F(x^k)$、$\nabla F(x^k)$、$\varphi(x^k)$ 及 $\nabla\varphi(x^k)$，令 $j=0$；其中 $\varphi(x)=\frac{1}{2}F(x)^T F(x)$；

❸ 解方程组 $[\nabla F(x^k)^T \nabla F(x^k)+\mu_k I]p_k=-\nabla\varphi(x^k)$；

❹ 计算 $x^{k+1}=x^k+p_k$ 及 $\varphi(x^{k+1})$；

❺ 如果 $\varphi(x^{k+1})<\varphi(x^k)$ 且 $j=0$，则取 $\mu_k=\frac{\mu_k}{v}$，$j=1$，转❸；

否则 $j\neq 0$，则转❼；

❻ 如果 $\varphi(x^{k+1})\geqslant\varphi(x^k)$，则取 $\mu_k=\mu_k v$，$j=1$，转❸；

❼ 若$\|p_k\| \leqslant \varepsilon$，则得到方程组的根，否则令$x^k = x^{k+1}$，转❷。

在 MATLAB 中编程实现的阻尼最小二乘法的函数为：mulDamp

功能：用阻尼最小二乘法求非线性方程组的一组解

调用格式：[r,m]=mulDamp(F,x0,h,u,v,eps)

其中，F：方程组；

x0：方程组的初始解；

h：数值微分增量步；

u：阻尼因子；

v：缩放常数；

eps：解的精度；

r：求得的一组解；

m：迭代步数。

阻尼最小二乘法的 MATLAB 程序代码如下所示：

```
function [r,m]=mulDamp(F,x0,h,u,v,eps)
%方程组：F
%方程组的初始解：x0
%数值微分增量步：h
%阻尼因子：u
%缩放常数：v
%解的精度：eps
%求得的一组解：r
%迭代步数：m
format long;
if nargin==5
    eps=1.0e-6;
end
FI = transpose(F)*F/2;
n = length(x0);
x0 = transpose(x0);
m=1;
tol=1;
while tol>eps
    j = 0;
    fx = subs(F,findsym(F),x0);
    J = zeros(n,n);
    for i=1:n
        x1 = x0;
        x1(i) = x1(i)+h;
        afx = subs(F,findsym(F),x1);
        J(:,i) = (afx-fx)/h;
    end
    FIx = subs(FI,findsym(FI),x0);
    for i=1:n
```

```
            x2 = x0;
            x2(i) = x2(i)+h;
            gradFI(i,1) = (subs(FI,findsym(FI),x2)-FIx)/h;
        end
        s=0;
        while s==0
            A = transpose(J)*J+u*eye(n,n);
            p = -A\gradFI;
            r = x0 + p;
            FIr = subs(FI,findsym(FI),r);
            if FIr<FIx
                if j == 0
                    u = u/v;
                    j = 1;
                else
                    s=1;
                end
            else
                u = u*v;
                j = 1;
                if norm(r-x0)<eps
                    s=1;
                end
            end
        end
        x0 = r;
        tol = norm(p);
        m=m+1;
        if(m>100000)                                    %迭代步数控制
            disp('迭代步数太多，可能不收敛！');
            return;
        end
    end
    format short;
```

例 10-18　阻尼最小二乘法解非线性方程组应用实例。用阻尼最小二乘法求下面方程组的一组解。

$$\begin{cases} x^2+y-6=0 \\ y^2-x-2=0 \end{cases}$$

初始迭代值取(4,4)。

解： 在 MATLAB 命令窗口中输入：

```
>>syms x y;
>> z=[x^2+y-6;y^2-x-2];
>> [r,m]=mulDamp(z,[4 4],0.1,8,2)
r =  1.9536
     1.9589
m =  5
```

用阻尼最小二乘法求下面方程组的解关键是取好阻尼因子，取得好的话，就可很快求得方程组的解，一定要注意$\upsilon>1$，否则结果不收敛。

10.17 小结

从本章介绍的几种数值方法可以看出，非线性方程组的数值解法本质上都是属于迭代法，而且每种迭代法不可避免地要求一个雅可比矩阵的逆，虽然可以求出任何一个非线性方程组的雅可比矩阵，但是其逆却不一定存在，因此那些直接求逆的迭代法是有一定局限性的，而有些迭代法却绕过了求逆的困难，如 D-F-P 算法和 B-F-S 算法等，这样它们的适用范围大一些，但也存在迭代时间过长的问题，因此要根据实际问题采用合适的算法。

第 11 章　解线性方程组的直接法

线性方程组解法的发展和计算机的出现有着莫大的关系，在计算机未出现前，线性方程组的解法是手工进行的，因此数值方法的发展受到了很大限制，但是随着计算机的发展，新的算法不断涌现，到现在还有不少人在研究线性方程组的解法，因为非线性方程组、微分方程的数值解法最终都要转化成线性方程组来求解。本章讨论线性方程组的直接解法，其迭代解法将在下一章中讨论。

通过本章的学习，读者不仅能掌握常见的解线性方程组的直接法，而且还能熟练使用 MATLAB 编程来实现这些算法。

11.1 线性方程组概论

设 A 为 $m\times n$ 的矩阵，x 为 $n\times 1$ 的列向量，b 为 $m\times 1$ 的列向量，则一般的线性方程组可表示为以下的形式：

$$Ax=b$$

利用线性代数的知识可以知道，如果 $m=n$，且矩阵 A 为满秩，则上面的线性方程组有唯一解，这就是本章所有算法的前提条件。

11.2 高斯消去法

高斯消去法是求解线性方程组最基本的方法，也是最原始的方法。由于此方法简单，计算量小，因此解决一般的线性方程组的主流方法也是高斯消去法，同时它也常常用来求矩阵的逆。

高斯消去法的基本思路是先把系数矩阵通过行变换转化为上三角矩阵，然后再求解变换后的上三角形线性方程组，即回代过程。为了后面的程序编写简单，将上三角系数矩阵和下三角系数矩阵的线性方程组的求解过程编成函数如下。

在 MATLAB 中编程实现的上三角系数矩阵求解函数为：SolveUpTriangle

功能：求上三角系数矩阵的线性方程组 Ax=b 的解

调用格式：x= SolveUpTriangle (A,b)

其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；

x：线性方程组的解。

上三角系数矩阵求解函数用 MATLAB 实现如下所示：

```
function x=SolveUpTriangle(A,b)
%求上三角系数矩阵的线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%线性方程组的解：x
N = size(A);
n = N(1);
for i=n:-1:1
    if(i<n)
        s = A(i,(i+1):n)*x((i+1):n,1);
    else
        s = 0;
    end
    x(i,1)=(b(i)-s)/A(i,i);
end
```

在 MATLAB 中编程实现的下三角系数矩阵求解函数为：SolveDownTriangle

功能：求下三角系数矩阵的线性方程组 Ax=b 的解

调用格式：x= SolveDownTriangle (A,b)

其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；

x：线性方程组的解。

下三角系数矩阵求解函数用 MATLAB 实现如下所示：

```
function x=SolveDownTriangle(A,b)
%求下三角系数矩阵的线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%线性方程组的解：x
N = size(A);
n = N(1);
for i=1:n
    if(i>1)
        s = A(i,1:(i-1))*x(1:(i-1),1);
    else
        s = 0;
    end
    x(i,1)=(b(i)-s)/A(i,i);
end
```

11.2.1 高斯顺序消去法

高斯顺序消去法的算法步骤介绍如下。

❶ 将方程组写成以下的增广矩阵的形式：

$$\begin{Bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{Bmatrix}$$

❷ 对 $k=1,2,3,\cdots,n-1$，对 $j=k+1,k+2,\cdots,n$，计算

$$a_{jm}=a_{jm}-\frac{a_{km}\bullet a_{jk}}{a_{kk}} \qquad (m=k,k+1,\cdots,n)$$

$$b_j=b_j-\frac{b_k\bullet a_{jk}}{a_{kk}}$$

算法结束。

依此算法就可将系数矩阵变成上三角矩阵，这时可用回代过程求解。例如下面的过程就是高斯顺序消去法：

$$\begin{bmatrix} 1 & 0 & -2 & 1 \\ 3 & 2 & -1 & 1 \\ -5 & 4 & 6 & 1 \end{bmatrix} \to \begin{bmatrix} 1 & 0 & -2 & 1 \\ 0 & 2 & 5 & -2 \\ 0 & 4 & -4 & 6 \end{bmatrix} \to \begin{bmatrix} 1 & 0 & -2 & 1 \\ 0 & 2 & 5 & -2 \\ 0 & 0 & -14 & 10 \end{bmatrix}$$

在 MATLAB 中编程实现的高斯顺序消去法函数为：GaussXQByOrder

功能：高斯顺序消去法求线性方程组 Ax=b 的解

调用格式：[x,XA]=GaussXQByOrder(A,b)

其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；

x：线性方程组的解；

XA：消元后的系数矩阵（可选的输出参数）。

高斯顺序消去法用 MATLAB 实现如下所示：

```
function [x,XA]=GaussXQByOrder(A,b)
%高斯顺序消去法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%线性方程组的解：x
%消元后的系数矩阵：XA
N = size(A);
n = N(1);
for i=1:(n-1)
    for j=(i+1):n
        if(A(i,i)==0)
            disp('对角元素为 0！');              %防止对角元素为 0
            return;
        end
        l = A(j,i);
        m = A(i,i);
        A(j,1:n)=A(j,1:n)-l*A(i,1:n)/m;  %消元方程
        b(j)=b(j)-l*b(i)/m;
```

```
    end
end
x=SolveUpTriangle(A,b);                %通用的求上三角系数矩阵线性方程组的函数
XA = A;                                %消元后的系数矩阵
```

例 11-1 高斯顺序消去法解线性方程组应用实例。用高斯顺序消去法求下列线性方程组的解。

$$\begin{bmatrix} 1 & 5 & 0 \\ 3 & 9 & 4 \\ -8 & 6 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A=[1 3 0;3 9 4;-8 6 2];
>> b=[1;1;1];
>> [x,XA]= GaussXQByOrder(A,b)
```

输出计算结果为：

```
x =  -0.0204
      0.2041
     -0.1939
XA =  1.0000    5.0000         0
          0   -6.0000    4.0000
          0         0   32.6667
```

输出结果中的 XA 就是系数矩阵经过消元后变成的上三角矩阵，然后调用通用的解上三角系数矩阵线性方程组的函数就可回代得到解。

在高斯顺序消去法的实际操作过程中，往往还可以先将对角线的元素化为 1，再消去主对角线以下的元素，此过程称为标准化的高斯顺序消去法。读者可以参照高斯顺序消去法的代码自己写出相应的标准化高斯顺序消去法的 MATLAB 代码。

高斯顺序消去法的计算量是消去和回代两部分的计算量之和，对于求解 n 变元的线性方程组，高斯顺序消去法的计算量的量级为 $O(n^3/3)$。

11.2.2 高斯主元消去法

高斯顺序消去法有一个最大的缺点就是一旦对角元素是 0，它就进行不下去了，为了解决这个问题，就有了高斯主元消去法。

11.2.2.1 高斯按列主元消去法

如果在高斯顺序消去法消去过程进行到第 i 步时，先选取 $a_{ri}(i \le r \le n)$ 中（即第 i 列）绝对值最大的元素，设为第 j 行的元素 a_{ji}，把矩阵的第 i 行和第 j 行互换，这时 a_{ii} 变为 a_{ji}，然后将第 $i+1$ 行至第 n 行中的每一行减去第 i 行乘以 $\frac{a_{ki}}{a_{ii}}$（k 代表行号），依此进行消元，这样得到的算法叫高斯按列主元消去法。

高斯主元消去法的算法步骤介绍如下。

❶ 将方程组写成以下的增广矩阵的形式：

$$\begin{Bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{Bmatrix}$$

❷ 对 $k=1,2,3,\cdots,n-1$，令 $a_{pk}=\max\sum_{s=k}^{n}|a_{sk}|$；交换增广矩阵的第 k 行与第 p 行；对 $j=k+1,k+2,\cdots,n$，计算

$$a_{jm}=a_{jm}-\frac{a_{km}\bullet a_{jk}}{a_{kk}} \quad (m=k,k+1,\cdots,n)$$

$$b_j=b_j-\frac{b_k\bullet a_{jk}}{a_{kk}}$$

算法结束。

例如下面的消去过程：

$$\begin{bmatrix} 3 & 0 & 5 & 1 \\ 6 & 4 & 2 & 2 \\ 2 & 7 & 10 & 3 \end{bmatrix} \to \begin{bmatrix} 6 & 4 & 2 & 2 \\ 3 & 0 & 5 & 1 \\ 2 & 7 & 10 & 3 \end{bmatrix} \to \begin{bmatrix} 6 & 4 & 2 & 2 \\ 0 & -2 & 4 & 0 \\ 0 & 17/3 & 28/3 & 7/3 \end{bmatrix} \to$$

$$\begin{bmatrix} 6 & 4 & 2 & 2 \\ 0 & 17/3 & 28/3 & 7/3 \\ 0 & -2 & 4 & 0 \end{bmatrix} \to \begin{bmatrix} 6 & 4 & 2 & 2 \\ 0 & 17/3 & 28/3 & 7/3 \\ 0 & 0 & 124/17 & 14/17 \end{bmatrix}$$

在 MATLAB 中编程实现的高斯按列主元消去法函数为：GaussXQLineMain

功能：高斯按列主元消去法求线性方程组 Ax=b 的解

调用格式：[x,XA]= GaussXQLineMain (A,b)

其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；

x：线性方程组的解；

XA：消元后的系数矩阵（可选的输出参数）。

高斯按列主元消去法用 MATLAB 实现如下所示：

```
function [x,XA]=GaussXQLineMain(A,b)
%高斯按列主元消去法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%线性方程组的解：x
%消元后的系数矩阵：XA
N = size(A);
n = N(1);
index = 0;
for i=1:(n-1)
```

```
    me = max(abs(A(1:n,i)));        %选取列主元
    for k=i:n
        if(abs(A(k,i))==me)
            index = k;              %保存列主元所在的行
            break;
        end
    end
    temp = A(i,1:n);
    A(i,1:n) = A(index,1:n);
    A(index,1:n) = temp;
    bb = b(index);
    b(index)=b(i);
    b(i) = bb;                      %交换主行
    for j=(i+1):n
        if(A(i,i)==0)
            disp('对角元素为 0! ');
            return;
        end
        l = A(j,i);
        m = A(i,i);
        A(j,1:n)=A(j,1:n)-l*A(i,1:n)/m;
        b(j)=b(j)-l*b(i)/m;         %消元
    end
end
x=SolveUpTriangle(A,b);
XA = A;
```

例 11-2 高斯按列主元消去法解线性方程组应用实例。用高斯按列主元消去法求下列线性方程组的解。

$$\begin{bmatrix} 3 & 6 & 0 \\ 7 & -2 & 5 \\ 2 & 1 & 8 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A=[3 6 0;7 -2 5;2 1 8];
>> b=[1;1;1];
>> [x,XA]= GaussXQLineMain(A,b)
```

输出计算结果为：

```
x =  0.1150
     0.1091
     0.0826
XA =  7.0000   -2.0000    5.0000
           0    6.8571   -2.1429
           0         0    7.0625
```

从 XA 第一行的第一个元素为 7 可以看出，第二行与第一行已经交换了，如果想知道选主元的每一步过程，读者可以自己修改程序，把每一步的系数矩阵打印出来。

11.2.2.2 高斯全主元消去法

如果在高斯顺序消去法的消去过程进行到第 i 步时，先选取 $a_{rs}(i \le r \le n, i \le s \le n)$ 中绝对值最大的元素，设为 $a_{r_i s_i}$，把矩阵的第 i 行和第 r_i 行互换，第 i 列和第 s_i 列互换，这时 a_{ii} 变为 $a_{r_i s_i}$，然后将第 $i+1$ 行至第 n 行中的每一行减去第 i 行乘以 $\frac{a_{ki}}{a_{ii}}$（k 代表行号），依此进行消元，这样得到的算法叫做高斯全主元消去法。

高斯全主元消去法的算法步骤介绍如下。

❶ 将方程组写成以下的增广矩阵的形式：

$$\begin{Bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{Bmatrix}$$

❷ 对 $k=1,2,3,\cdots,n-1$，令 $a_{pq} = \max \sum\limits_{\substack{k \le s \le n \\ k \le t \le n}} |a_{st}|$；交换增广矩阵的第 k 行与第 p 行；交换增广矩阵的第 k 列与第 q 列；对 $j=k+1,k+2,\cdots,n$，计算

$$a_{jm} = a_{jm} - \frac{a_{km} \cdot a_{jk}}{a_{kk}} \quad (m=k,k+1,\cdots,n)$$

$$b_j = b_j - \frac{b_k \cdot a_{jk}}{a_{kk}}$$

算法结束。

例如下面的消去过程：

$$\begin{bmatrix} 3 & 0 & 15 & 1 \\ 6 & 4 & 2 & 2 \\ 10 & 7 & 10 & 3 \end{bmatrix} \to \begin{bmatrix} 15 & 0 & 3 & 1 \\ 2 & 4 & 6 & 2 \\ 10 & 7 & 10 & 3 \end{bmatrix} \to \begin{bmatrix} 15 & 0 & 3 & 1 \\ 0 & 4 & 28/5 & 28/15 \\ 0 & 7 & 8 & 7/3 \end{bmatrix} \to$$

$$\begin{bmatrix} 15 & 0 & 3 & 1 \\ 0 & 7 & 8 & 7/3 \\ 0 & 4 & 28/5 & 28/15 \end{bmatrix} \to \begin{bmatrix} 15 & 0 & 3 & 1 \\ 0 & 8 & 7 & 7/3 \\ 0 & 28/5 & 4 & 28/15 \end{bmatrix} \to \begin{bmatrix} 15 & 0 & 3 & 1 \\ 0 & 8 & 7 & 7/3 \\ 0 & 0 & -9/10 & 7/30 \end{bmatrix}$$

在 MATLAB 中编程实现的高斯全主元消去法函数为：GaussXQAllMain

功能：高斯全主元消去法求线性方程组 Ax=b 的解

调用格式：[x,XA]= GaussXQAllMain (A,b)

其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；

x：线性方程组的解；

XA：消元后的系数矩阵（可选的输出参数）；

高斯全主元消去法用 MATLAB 实现如下所示：

```
function [x,XA]= GaussXQAllMain (A,b)
```

```
%高斯全主元消去法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵: A
%线性方程组中的常数向量: b
%线性方程组的解: x
%消元后的系数矩阵: XA
N = size(A);
n = N(1);
index_l = 0;
index_r = 0;
order = 1:n;                              %记录未知数顺序的向量
for i=1:(n-1)
    me = max(max(abs(A(i:n,i:n))));    %选取全主元
    for k=i:n
        for r=i:n
            if(abs(A(k,r))==me)
                index_l = k;
                index_r = r;      %保存主元所在的行和列
                k=n;
                break;
            end
        end
     end
     temp = A(i,1:n);
     A(i,1:n) =  A(index_l,1:n);
     A(index_l,1:n) = temp;
     bb = b(index_l);
     b(index_l)=b(i);
     b(i) = bb;                %交换主行
     temp = A(1:n,i);
     A(1:n,i) = A(1:n,index_r);
     A(1:n,index_r) = temp;       %交换主列
     pos = order(i);
     order(i) = order(index_r);
     order(index_r) = pos;        %主列的交换会造成未知数顺序的变化
     for j=(i+1):n
        if(A(i,i)==0)
           disp('对角元素为 0! ');
           return;
        end
        l = A(j,i);
        m = A(i,i);
        A(j,1:n)=A(j,1:n)-l*A(i,1:n)/m;
        b(j)=b(j)-l*b(i)/m;
    end
end
x=SolveUpTriangle(A,b);
y=zeros(n,1);
for i=1:n
    for j=1:n
        if(order(j)==i)
```

```
            y(i)=x(j);
        end
    end
end                         %恢复未知数原来的顺序
x=y;
XA = A;
```

例 11-3　高斯全主元消去法解线性方程组应用实例。用高斯全主元消去法求下列线性方程组的解。

$$\begin{bmatrix} 10 & 6 & 4 \\ 5 & 0 & 1 \\ 2 & -1 & 8 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解： 在 MATLAB 命令窗口中输入：

```
>> A=[10 6 4;5 0 1;2 -1 8];
>> b=[1;1;1];
>> [x,XA]= GaussXQAllMain (A,b)
```

输出计算结果为：

```
x =  0.1891
    -0.1849
     0.0546
XA =  10.0000    4.0000    6.0000
           0     7.2000   -2.2000
           0          0   -3.3056
```

高斯全主元消去法的想法很好，但是用程序实现起来却并不简单，很容易犯错误，因为在消去过程中进行了列交换，将未知数的顺序打乱了，因此在每次列交换后要记下未知数的排列顺序，最后要调整回来。否则得出的结果数值是对的，但顺序却是错的。

11.2.3　高斯-若当消去法

高斯-若当消去法不需要进行行或列的交换，而且它的消元比较彻底，每行只留下一个非零元素。

高斯-若当消去法的算法步骤介绍如下。

❶ 将方程组写成以下的增广矩阵的形式：

$$\begin{Bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{Bmatrix}$$

建立一个数组 $pos[n]$，将其初始化为 0，用以保存每列的主元；

❷ 对 $k=1,2,3,\cdots,n$ ，令 $a_{pk}=\max \sum\limits_{\substack{1\le s\le n \\ s\ne pos[1],\cdots,pos[k-1]}} |a_{sk}|$ ； $pos[k]=p$ ； 对

$j=1,2,\cdots,p-1,p+1,\cdots,n$ ，计算

$$a_{jm}=a_{jm}-\frac{a_{pm}\cdot a_{jk}}{a_{pk}} \quad (m=k,k+1,\cdots,n)$$

$$b_j=b_j-\frac{b_p\cdot a_{jk}}{a_{pk}}$$

算法结束。

例如下面的消去过程：

$$\begin{bmatrix}3&8&5&1\\6&4&2&1\\2&1&10&1\end{bmatrix}\to\begin{bmatrix}0&6&4&1/2\\6&4&2&1\\0&-1/3&28/3&2/3\end{bmatrix}\to\begin{bmatrix}0&6&4&1/2\\6&0&-2/3&2/3\\0&0&86/9&25/36\end{bmatrix}\to$$

$$\begin{bmatrix}0&6&0&9/43\\6&0&0&123/172\\0&0&86/9&25/36\end{bmatrix}$$

在 MATLAB 中编程实现的高斯-若当消去法函数为：GaussJordanXQ

功能：高斯-若当消去法求线性方程组 Ax=b 的解

调用格式：[x,XA]= GaussJordanXQ (A,b)

其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；

x：线性方程组的解；

XA：消元后的系数矩阵（可选的输出参数）。

高斯-若当消去法用 MATLAB 实现如下所示：

```
function [x,XA]= GaussJordanXQ (A,b)
%高斯-若当消去法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%线性方程组的解：x
%消元后的系数矩阵：XA
N = size(A);
n = N(1);
index = 0;
pos = zeros(n,1);
B = A;
for i=1:n
    me = max(abs(B(1:n,i)));          %选取列主元
    for k=1:n
        if(abs(A(k,i))==me)
            index = k;
            pos(i,1) = k;           %保存列主元所在的行号
            break;
        end
    end
```

```
        m = A(index,i);
        for j=1:n
            if(j ~= index)
                l = A(j,i);
                A(j,1:n)=A(j,1:n)-l*A(index,1:n)/m;
                b(j)=b(j)-l*b(index)/m;    %消元
            end
        end
        B = A;
        for k=1:n
            if(pos(k,1)~=0)
                B(pos(k,1),1:n)=0;         %避免列主元在同一行
            end
        end
    end
    XA = A;
    for i=1:n
        x(i,1)=b(pos(i,1))/A(pos(i,1),i);  %求解
    end
```

例 11-4 高斯-若当消去法解线性方程组应用实例。用高斯-若当消去法求下列线性方程组的解。

$$\begin{bmatrix} 1 & 3 & 8 \\ -5 & 2 & 9 \\ 0 & 1 & 4 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A=[1 3 8;-5 2 9;0 1 4];
>> b=[1;1;1];
>> [x,XA]= GaussJordanXQ(A,b)
```

输出计算结果为：

```
x =  0.3158
    -1.3158
     0.5789
XA =        0      3.4000         0
     -5.0000           0          0
           0           0     1.1176
```

高斯消去法十分简单，只不过要注意避免列主元在同一行，即每次选取的主元必须在不同行。

11.3 三角分解法

如果系数矩阵 A 可以分解为两个三角矩阵的乘积：

$$A = LU$$

其中，L 为下三角矩阵，U 为上三角矩阵。那么线性方程组 Ax = b 的求解可分为以下的两步：

第一步，求解方程组 $Ly = b$ ，解出 y；

第二步，求解方程组 $Ux = y$ ，解出 x；

而求解这两个三角形方程组是很简单的。按 L 和 U 形式的不同，三角分解法可分为克劳特分解法和多利特勒分解法。

11.3.1 克劳特分解法

在克劳特分解法中，L 和 U 的形式如下所示：

$$L=\begin{bmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & \cdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix}, U=\begin{bmatrix} 1 & u_{12} & \cdots & u_{1n} \\ & 1 & \cdots & u_{2n} \\ & & \ddots & \vdots \\ & & & 1 \end{bmatrix}$$

克劳特分解的算法步骤介绍如下。

❶ 计算 L 的第一列和 U 的第一行元素：

$$l_{i1} = a_{i1} \qquad i = 1,2,\cdots,n;$$
$$u_{1j} = a_{1j} / l_{11} \qquad j = 2,\cdots,n$$

❷ 对 $k = 2,\cdots,n$ ，计算

$$l_{ik} = a_{ik} - \sum_{p=1}^{k-1} l_{ip}u_{pk} \qquad i = k,\cdots,n$$

$$u_{kj} = (a_{kj} - \sum_{s=1}^{k-1} l_{ks}u_{sj}) / l_{kk} \qquad j = k+1,\cdots,n$$

❸ 解三角形方程组：

$$\begin{cases} y_1 = b_1 / l_{11} \\ y_k = (b_k - \sum_{t=1}^{k-1} l_{kt}y_t) / l_{kk} \qquad k = 2,\cdots,n. \end{cases}$$

$$\begin{cases} x_n = y_n \\ x_k = y_k - \sum_{r=k+1}^{n} u_{kr}x_r \qquad k = n-1,\cdots,1. \end{cases}$$

在 MATLAB 中编程实现的克劳特分解法函数为：Crout

功能：克劳特分解法求线性方程组 Ax=b 的解

调用格式：[x,L,U]=Crout(A,b)

其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；

x：线性方程组的解；

L：克劳特分解后的 L（可选的输出参数）；

U：克劳特分解后的 U（可选的输出参数）。

克劳特分解法用 MATLAB 实现如下所示：

```
function [x,L,U]=Crout(A,b)
%克劳特分解法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%线性方程组的解：x
%克劳特分解后的下三角矩阵：L
%克劳特分解后的上三角矩阵：U
N = size(A);
n = N(1);
L = zeros(n,n);
U = eye(n,n);                %U 的对角元素为 1
L(1:n,1) = A(1:n,1);          %L 的第一列
U(1,1:n) = A(1,1:n)/L(1,1);  %U 的第一行
for k=2:n
    for i=k:n
        L(i,k) = A(i,k)-L(i,1:(k-1))*U(1:(k-1),k);
        %L 的第 k 列
    end
    for j=(k+1):n
        U(k,j) = (A(k,j)-L(k,1:(k-1))*U(1:(k-1),j))/L(k,k);
        %U 的第 k 行
    end
end
y = SolveDownTriangle(L,b);
x = SolveUpTriangle(U,y);        %求解方程
```

例 11-5 克劳特分解法解线性方程组应用实例。用克劳特分解法求下列线性方程组的解。

$$\begin{bmatrix} 6 & 3 & -8 \\ 15 & 5 & 2 \\ 2 & 0 & 7 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解： 在 MATLAB 命令窗口中输入：

```
>> A=[6 3 -8;15 5 2;2 0 7];
>> b=[1;1;1];
>> [x,L,U]=Crout(A,b)
```

输出计算结果为：

```
x =  -4.6154
     13.4615
      1.4615
L =  6.0000         0          0
```

```
        15.0000   -2.5000         0
         2.0000   -1.0000    0.8667
    U =  1.0000    0.5000   -1.3333
              0    1.0000   -8.8000
              0         0    1.0000
```

11.3.2 多利特勒分解法

多利特勒分解法中，L 和 U 的形式如下所示：

$$L=\begin{bmatrix}1 & & & \\ l_{21} & 1 & & \\ \vdots & \cdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & 1\end{bmatrix},U=\begin{bmatrix}u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & \cdots & u_{2n} \\ & & \ddots & \vdots \\ & & & u_{nn}\end{bmatrix}$$

多利特勒分解的算法步骤介绍如下。

对 $k=2,\cdots,n$，

❶ 计算

$$u_{kj}=a_{kj}-\sum_{m=1}^{k-1}l_{km}u_{mj} \qquad j=k,k+1,\cdots,n.$$

❷ 计算

$$l_{ik}=(a_{ik}-\sum_{m=1}^{k-1}l_{im}u_{mk})/u_{kk} \qquad i=k+1,\cdots,n$$

❸ 解三角形方程组：

$$\begin{cases}y_1=b_1 \\ y_i=(b_i-\sum_{t=1}^{k-1}l_{it}y_t) \qquad i=2,\cdots,n.\end{cases}$$

$$\begin{cases}x_n=y_n/u_{nn} \\ x_i=(y_i-\sum_{r=k+1}^{n}u_{ir}x_r)/u_{ii} \qquad i=n-1,\cdots,1.\end{cases}$$

> 在 MATLAB 中编程实现的多利特勒分解法函数为：Doolittle
>
> 功能：多利特勒分解法求线性方程组 Ax=b 的解
>
> 调用格式：[x,L,U]= Doolittle (A,b)
>
> 其中，A：线性方程组的系数矩阵；
>
> b：线性方程组中的常数向量；
>
> x：线性方程组的解；
>
> L：多利特勒分解后的 L（可选的输出参数）；
>
> U：多利特勒分解后的 U（可选的输出参数）。

多利特勒分解法用 MATLAB 实现如下所示：

```
function [x,L,U]= Doolittle (A,b)
%多利特勒分解法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%线性方程组的解：x
%多利特勒分解后的下三角矩阵：L
%多利特勒分解后的上三角矩阵：U
N = size(A);
n = N(1);
L = eye(n,n);              %L 的对角元素为 1
U = zeros(n,n);
U(1,1:n) = A(1,1:n);        %U 的第一行
L(1:n,1) = A(1:n,1)/U(1,1); %L 的第一列
for k=2:n
    for i=k:n
        U(k,i) = A(k,i)-L(k,1:(k-1))*U(1:(k-1),i);
        %U 的第 k 行
    end
    for j=(k+1):n
        L(j,k) = (A(j,k)-L(j,1:(k-1))*U(1:(k-1),k))/U(k,k);
        %L 的第 k 列
    end
end
y = SolveDownTriangle(L,b);
x = SolveUpTriangle(U,y);     %求解方程
```

例 11-6 多利特勒分解法解线性方程组应用实例。用多利特勒分解法求下列线性方程组的解。

$$\begin{bmatrix} 6 & 3 & -8 \\ 15 & 5 & 2 \\ 2 & 0 & 7 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解： 在 MATLAB 命令窗口中输入：

```
>> A=[6 3 -8;15 5 2;2 0 7];
>> b=[1;1;1];
>> [x,L,U]= Doolittle(A,b)
```

输出计算结果为：

```
x =  -4.6154
     13.4615
      1.4615
L =  1.0000        0          0
     2.5000    1.0000         0
     0.3333    0.4000    1.0000
U =  6.0000    3.0000   -8.0000
         0   -2.5000   22.0000
         0         0    0.8667
```

得出的 L 和 U 和上题相比发生了变化。多利特勒分解法的运算量和克劳特分解法是一样的。如果矩阵的第一行第一列的元素为 0，不管是克劳特分解法，还是多利特勒分解法，都会出现被除数为 0 的错误，这是三角分解法最大的弊端，当然也有解决的办法，那就是在三角分解之前，检查矩阵第一行第一列的元素是否为 0，如果是，通过行交换，将第一列中不为 0 的元素交换到第一行，再进行三角分解，可参考下面的代码段：

```
……
if(A(1,1)==0)
    me = max(abs(A(1:n,i)));          %选取列主元
   for k=i:n
       if(abs(A(k,i))==me)
         index = k;                   %保存列主元所在的行
         break;
       end
    end
    temp = A(i,1:n);
    A(i,1:n) =  A(index,1:n);
    A(index,1:n) = temp;
    bb = b(index);
    b(index)=b(i);
    b(i) = bb;                        %交换主行
end
……
```

11.4 乔列斯基分解法

乔列斯基分解法来源于三角分解法，将三角分解法应用于对称正定矩阵就得到乔列斯基分解法，具体又可分为三种相似的方法。

11.4.1 对称正定矩阵的 LL^T 分解法

在三角分解法中，令 $U = L^T$ 就得到对称正定矩阵的 LL^T 分解法。其算法步骤介绍如下。

❶ 对于 $k = 1,2,\cdots,n$，计算：

$$l_{11} = \sqrt{a_{11}}, l_{i1} = a_{i1} / l_{11} \qquad (i = 2,3,\cdots,n)$$

$$l_{kk} = \sqrt{a_{kk} - \sum_{m=1}^{k-1} l_{km}^2}$$

$$l_{ik} = (a_{ik} - \sum_{m=1}^{k-1} l_{im} l_{km}) / l_{kk} \quad (i = k+1,\cdots,n)$$

❷ 解三角形方程组。

在 MATLAB 中编程实现的 LL^T 分解法函数为：SymPos1

功能：LL^T 分解法求线性方程组 Ax=b 的解

调用格式：[x,L]= SymPos1 (A,b)
其中，A：线性方程组的系数矩阵；
　　b：线性方程组中的常数向量；
　　x：线性方程组的解；
　　L：LL^T 分解后的 L（可选的输出参数）。

LL^T 分解法用 MATLAB 实现如下所示：

```
function [x,L]= SymPos1 (A,b)
%LL^T 分解法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%线性方程组的解：x
%LL^T 分解后的下三角矩阵：L
N = size(A);
n = N(1);
L(1,1) = sqrt(A(1,1));
L(2:n,1) = A(2:n,1)/L(1,1);   %L 的第一列
for k=2:n
   L(k,k) = sqrt(A(k,k)-L(k,1:(k-1))*transpose(L(k,1:(k-1))));%L 对角元素
  for i=(k+1):n
   L(i,k) = (A(i,k)-L(i,1:(k-1))*transpose(L(k,1:(k-1))))/L(k,k);%L 第 k 列
  end
end
y = SolveDownTriangle(L,b);
x = SolveUpTriangle(transpose(L),y);   %求解方程
```

例 11-7　对称正定矩阵的 LL^T 分解法解线性方程组应用实例。用 LL^T 分解法求下列线性方程组的解。

$$\begin{bmatrix} 6 & 3 & -2 \\ 3 & 5 & 1 \\ -2 & 1 & 7 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A=[6 3 -2;3 5 1;-2 1 7];
>> b=[1;1;1];
>> [x,L]= SymPos1(A,b)
```

输出计算结果为：

```
x =  0.2202
     0.0275
     0.2018
L =  2.4495        0          0
     1.2247    1.8708         0
    -0.8165    1.0690    2.2783
```

当然，要得出上面的正确结果，首先得保证系数矩阵对称正定才行。

11.4.2 对称正定矩阵的 LDL^T 分解法

在上面的分解中，需要开方运算，这是比较不利的。为了避免开方运算，令 L 的对角元素都为 1，这便得到 LDL^T 分解法，其中 L 为下三角阵，D 为对角阵。其算法步骤介绍如下：

❶ 对于 $k=1,2,\cdots,n,$，计算：

$$l_{kk}=1,$$

$$d_k=a_{kk}-\sum_{m=1}^{k-1}l_{km}^2 d_m,$$

$$l_{ik}=(a_{ik}-\sum_{m=1}^{k-1}l_{im}d_m l_{km})/d_k\,(i=k+1,\cdots,n)$$

❷ 解三角形方程组：

$$\begin{cases}Ly=b\\ L^T x=D^{-1}y\end{cases}$$

在 MATLAB 中编程实现的 LDL^T 分解法函数为：SymPos2

功能：LDL^T 分解法求线性方程组 Ax=b 的解

调用格式：[x,L,D]= SymPos2 (A,b)

其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；

x：线性方程组的解；

L：LDL^T 分解后的 L（可选的输出参数）；

D：LDL^T 分解后的 D（可选的输出参数）。

LDL^T 分解法用 MATLAB 实现如下所示：

```
function [x,L,D]= SymPos2 (A,b)
%LDL^T 分解法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%线性方程组的解：x
%LDL^T 分解后的下三角矩阵：L
%LDL^T 分解后的 d 对角矩阵：D
N = size(A);
n = N(1);
L = eye(n,n);       %L 的对角元素为 1
d = zeros(n,1);
for k=1:n
    d(k,1) = A(k,k)-L(k,1:(k-1)).*L(k,1:(k-1))*d(1:(k-1),1);
    %D 的第 k 个对角元素
    for i=(k+1):n
```

```
        L(i,k) = (A(i,k)-L(i,1:(k-1)).*L(k,1:(k-1))*d(1:(k-1),1))/d(k,1);
        %L 的第 k 列
    end
end
D = diag(d);
y = SolveDownTriangle(L,b);      %求解 y
for i=1:n
   y(i,1)=y(i,1)/d(i,1);
end
x = SolveUpTriangle(transpose(L),y);  %求解 x
```

例 11-8　对称正定矩阵的 LDL^T 分解法解线性方程组应用实例。用 LDL^T 分解法求下列线性方程组的解。

$$\begin{bmatrix} 6 & 3 & -2 \\ 3 & 5 & 1 \\ -2 & 1 & 7 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A=[6 3 -2;3 5 1;-2 1 7];
>> b=[1;1;1];
>> [x,L,D]= SymPos2(A,b)
```

输出计算结果为：

```
x =  0.2202
     0.0275
     0.2018
L =  1.0000         0         0
     0.5000    1.0000         0
    -0.3333    0.5714    1.0000
D =  6.0000         0         0
          0    3.5000         0
          0         0    5.1905
```

要注意的是 D 矩阵对角线上的元素并不是系数矩阵的特征值，就拿上例来说，矩阵 A 的特征值为 1.6515、7.2698 与 9.0787。

11.4.3　对称正定矩阵的改进 LDL^T 分解法

虽然 LDL^T 分解法避开了开方运算，但是它增加了乘法运算量，为了减少运算量，就有如下的改进算法。其算法步骤介绍如下。

❶ 对 $k=1,2,\cdots,n$，计算：

$$d_k = a_{kk} - \sum_{m=1}^{k-1} \hat{a}_{km} l_{km}$$

$$\hat{a}_{ik} = a_{ik} - \sum_{m=1}^{k-1} \hat{a}_{im} l_{km} \quad (i = k+1,\cdots,n)$$

$$l_{ik} = \hat{a}_{ik} / d_k$$

❷ 解三角形方程组：

$$\begin{cases} Ly = b \\ L^T x = D^{-1} y \end{cases}$$

在 MATLAB 中编程实现改进的 LDL^T 分解法函数为：SymPos3

功能：改进的 LDL^T 分解法求线性方程组 Ax=b 的解

调用格式：[x,L,D]= SymPos3 (A,b)

其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；

x：线性方程组的解；

L：改进的 LDL^T 分解后的 L（可选的输出参数）；

D：改进的 LDL^T 分解后的 D（可选的输出参数）。

改进的 LDL^T 分解法用 MATLAB 实现如下所示：

```
function [x,L,D]= SymPos3 (A,b)
%改进的 LDL^T 分解法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%线性方程组的解：x
%改进的 LDL^T 分解后的下三角矩阵：L
%改进的 LDL^T 分解后的 d 对角矩阵：D
N = size(A);
n = N(1);
L = eye(n,n);              %L 的对角元素为 1
d = zeros(n,1);
AA = zeros(n,n);           %保存的中间变量
for k=1:n
   d(k,1) = A(k,k)-AA(k,1:(k-1))*transpose(L(k,1:(k-1)));
   %D 的第 k 个对角元素
   for i=(k+1):n
      AA(i,k) = A(i,k)-AA(i,1:(k-1))*transpose(L(k,1:(k-1)));
      %AA 的第 k 列
      L(i,k) = AA(i,k)/d(k,1);
      %L 的第 k 列
   end
end
D = diag(d);
y = SolveDownTriangle(L,b);                    %求解 y
for i=1:n
   y(i,1)=y(i,1)/d(i,1);
end
x = SolveUpTriangle(transpose(L),y);           %求解 x
```

例 11-9 对称正定矩阵的改进 LDL^T 分解法解线性方程组应用实例。用改进的 LDL^T

分解法求下列线性方程组的解。

$$\begin{bmatrix} 6 & 3 & -2 \\ 3 & 5 & 1 \\ -2 & 1 & 7 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解： 在 MATLAB 命令窗口中输入：

```
>> A=[6 3 -2;3 5 1;-2 1 7];
>> b=[1;1;1];
>> [x,L,D]= SymPos3(A,b)
```

输出计算结果为：

```
x =  0.2202
     0.0275
     0.2018
L =  1.0000         0         0
     0.5000    1.0000         0
    -0.3333    0.5714    1.0000
D =  6.0000         0         0
          0    3.5000         0
          0         0    5.1905
```

虽然改进的 LDL^T 分解法减少了乘法的运算量，但它引入了一个中间变量，增大了存储量，有利也有弊。

上面的三种方法是针对系数矩阵为对称正定矩阵的线性方程组，至于孰优孰劣，就看读者关注的是时间还是存储空间，如果存储空间足够大，显然第三种方法是首选方法。

11.5 三对角方程组的追赶法

对于系数矩阵为对角占优的三对角矩阵，它的三角分解比较特殊，如下所示：

$$\begin{bmatrix} a_1 & d_1 & & \\ c_1 & a_2 & \ddots & \\ & \ddots & \ddots & d_{n-1} \\ & & c_{n-1} & a_n \end{bmatrix}=\begin{bmatrix} 1 & & & \\ l_1 & 1 & & \\ & \ddots & \ddots & \\ & & l_{n-1} & 1 \end{bmatrix}\begin{bmatrix} u_1 & d_1 & & \\ & u_2 & \ddots & \\ & & \ddots & d_{n-1} \\ & & & u_n \end{bmatrix}$$

其中 L 为下三角的两对角矩阵，主对角线元素全为 1，而 U 为上三角的两对角矩阵，这时它的求解变得十分简单，本质上这是一种三角分解法，但习惯上称之为追赶法。其算法步骤介绍如下：

❶ 计算：

$$u_1 = a_1$$
$$l_{i-1} = c_i / u_{i-1}$$
$$u_i = a_i - l_{i-1}d_{i-1} \qquad i = 2,3,\cdots n$$

❷ 求解：

$$\begin{cases} y_1 = b_1 \\ y_i = d_i - l_{i-1} y_{i-1} & i = 2,3,\cdots,n \end{cases}$$

$$\begin{cases} x_n = y_n / u_n \\ x_i = (y_i - d_i x_{i+1}) / u_i & i = n-1, n-2, \cdots, 1 \end{cases}$$

在 MATLAB 中编程实现的追赶法函数为：followup

功能：追赶法求线性方程组 Ax=b 的解

调用格式：x= followup (A,b)

其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；

x：线性方程组的解。

追赶法用 MATLAB 实现如下所示：

```
function x=followup(A,b)
%采用追赶法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%线性方程组的解：x
n = rank(A);
for(i=1:n)
    if(A(i,i)==0)
        disp('Error: 对角有元素为 0! ');
        return;
    end
end;
d = ones(n,1);
a = ones(n-1,1);
c = ones(n-1);
for(i=1:n-1)
    a(i,1)=A(i+1,i);
    c(i,1)=A(i,i+1);
    d(i,1)=A(i,i);
end
d(n,1) = A(n,n);
%求解 Ly=b 的解 y，解保存在 b 中，
for(i=2:n)
    d(i,1)=d(i,1) - (a(i-1,1)/d(i-1,1))*c(i-1,1);
    b(i,1)=b(i,1) - (a(i-1,1)/d(i-1,1))*b(i-1,1);
end
%求解 Ux=y 的解 x，
x(n,1) = b(n,1)/d(n,1);
for(i=(n-1):-1:1)
    x(i,1) = (b(i,1)-c(i,1)*x(i+1,1))/d(i,1);
end
```

例 11-10 追赶法求线性方程组解的应用实例。用追赶法求下列线性方程组的解。

$$\begin{bmatrix} 1 & -3 & & & \\ 8 & 2 & 0 & & \\ & 6 & 3 & 7 & \\ & & 12 & 4 & 9 \\ & & & -4 & 5 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A=[1 -3 0 0 0;8 2 0 0 0;0 6 3 7 0;0 0 12 4 9;0 0 0 -4 5];
>> b=[1;1;1;1;1];
>> x= followup(A,b)
```

输出计算结果为：

```
x =  0.1923
   -0.2692
   -0.6923
    0.6703
    0.7363
```

追赶法的适用范围比较窄，只有系数矩阵是三对角阵的时候才能使用它，但是可以通过豪斯霍尔德变换将普通矩阵变换成三对角矩阵，有关这方面的知识可参考矩阵分析与应用方面的书。

11.6 直接求逆法

直接求逆法也是一种可以考虑的方法，它通过某种方法求出系数矩阵的逆，求解就很简单了。通常求逆矩阵的方法有初等变换法、高斯-若当消去法、分块矩阵法、消秩法等。

11.6.1 加边法求逆矩阵

加边法求逆矩阵本质上是一种分块求逆，其算法步骤介绍如下。

❶ 将系数矩阵按下面的形式分块：

$$A_n=\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1,n-1} & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2,n-1} & a_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-1} & a_{n-1,n} \\ \hline a_{n1} & a_{n2} & \cdots & a_{n,n-1} & a_{nn} \end{array}\right]=\begin{bmatrix} A_{n-1} & u_n \\ v_n & a_{nn} \end{bmatrix}$$

同时将 A 的逆矩阵也写成相应的分块形式：

$$A_n^{-1}=\begin{bmatrix} p_{n-1} & r_n \\ q_n & 1/a_n \end{bmatrix}$$

其中各个部分的计算公式如下：

$$\begin{cases} r_n = -\dfrac{A_{n-1}^{-1}u_n}{a_n} & a_n = a_{nn} - v_n A_{n-1}^{-1}u_n \\ q_n = -\dfrac{v_n A_{n-1}^{-1}}{a_n} & p_{n-1} = A_{n-1}^{-1} + \dfrac{A_{n-1}^{-1}u_n v_n A_{n-1}^{-1}}{a_n} \end{cases}$$

，其中 A_{n-1}^{-1} 为 A_{n-1} 的逆矩阵

❷ 对 $k = n-1, n-2, \cdots, 1$

如果 $k > 1$，将 A_k 递归重复上面的过程；否则令 $A_1^{-1} = \dfrac{1}{A(n,n)}$；

算法结束。

在 MATLAB 中编程实现的加边求逆法函数为：InvAddSide

功能：加边求逆法求线性方程组 Ax=b 的解

调用格式：[x,IA]= InvAddSide (A,b)

其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；

x：线性方程组的解；

IA：系数矩阵的逆矩阵（可选的输出参数）。

追赶法用 MATLAB 实现如下所示：

```
function [x,IA]=InvAddSide(A,b)
%加边求逆法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%线性方程组的解：x
%系数矩阵的逆矩阵：IA
IA = Inv(A);
x = IA*b;
%求逆矩阵的递归函数
function invA=Inv(A)
N = size(A);
n = N(1);
if(n == 1)
    invA = 1/A(n,n);                    %递归的终止条件
else
    u = A(1:(n-1),n);
    v = A(n,1:(n-1));
    iA = Inv(A(1:(n-1),1:(n-1)));       %递归公式
    An = A(n,n) - v*iA*u;
    r = -iA*u/An;
    p = iA + iA*u*v*iA/An;
    q = -v*iA/An;
    invA = [p r;q 1/An];                %逆矩阵
end
```

例 11-11 加边求逆法求线性方程组解的应用实例。用加边求逆法求下列线性方程组

的解。

$$\begin{bmatrix} 8 & 3 & -2 \\ 4 & 8 & -1 \\ 0 & 2 & 8 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解：在 MATLAB 命令窗口中输入：

```
>> A=[8 3 -2;4 8 -1;0 2 8];
>> b=[1;1;1];
>> [x,IA]=InvAddSide(A,b)
```

输出计算结果为：

```
x =  0.1226
    0.0769
    0.1058
IA =  0.1587   -0.0673    0.0313
    -0.0769    0.1538        0
     0.0192   -0.0385    0.1250
```

可以验证 A·IA 确实等于单位阵。加边法求逆的运算量比较大，过程中的乘法运算比较多，而且还用到了递归函数，这是很耗时间的。下面的叶尔索夫法相对简单一些。

11.6.2 叶尔索夫法求逆矩阵

如果系数矩阵 A 的各阶主子行列式不为零，可以构造如下的矩阵序列：

$A^{(0)},\overline{A}^{(1)},A^{(1)},\overline{A}^{(2)},A^{(2)},\cdots,\overline{A}^{(n)},A^{(n)}$，其中 $A^{(0)}=A-I$。$\overline{A}_{ij}^{(m)}$ 的递推公式如下所示：

$$\overline{A}_{ij}^{(m)}=\begin{cases} A_{ij}^{(m-1)} & i\neq m \\ 1 & i=m=j \\ 0 & i=m\neq j \end{cases}$$

而 $A_{ij}^{(m)}$ 的递推公式如下所示：

$$A_{ij}^{(m)}=\overline{A}_{ij}^{(m)}-\frac{\overline{A}_{im}^{(m)}\bullet A_{mj}^{(m-1)}}{1+A_{mm}^{(m-1)}}$$

由这两个递推公式计算 n 步，就可得到：

$$A^{(n)}=A^{-1}$$

此种求逆矩阵的方法称为叶尔索夫法。

叶尔索夫法求逆矩阵的算法步骤介绍如下。

❶ 令 $A^{(0)}=A-I$。

❷ 对 $m=1,2,\cdots,n$，计算：

$$\overline{A}_{ij}^{(m)}=\begin{cases} A_{ij}^{(m-1)} & i\neq m \\ 1 & i=m=j \\ 0 & i=m\neq j \end{cases}$$

$$A_{ij}^{(m)} = \overline{A}_{ij}^{(m)} - \frac{\overline{A}_{im}^{(m)} \bullet A_{mj}^{(m-1)}}{1 + A_{mm}^{(m-1)}}$$

算法结束。

在 MATLAB 中编程实现的叶尔索夫求逆法函数为：Yesf

功能：叶尔索夫求逆法求线性方程组 Ax=b 的解

调用格式：[x,IA]= Yesf (A,b)

其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；

x：线性方程组的解；

IA：系数矩阵的逆矩阵（可选的输出参数）；

追赶法用 MATLAB 实现如下所示：

```
function [x,IA]= Yesf(A,b)
%叶尔索夫求逆法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%线性方程组的解：x
%系数矩阵的逆矩阵：IA
N = size(A);
n = N(1);
A1 = A -eye(n,n);
A2 = zeros(n,n);
A3 = zeros(n,n);
for m=1:n
    for i=1:n
        for j=1:n
            if(i ~= m)
                A2(i,j) = A1(i,j);
            else
                if(j == m)
                    A2(i,j) = 1;
                else
                    A2(i,j) = 0;
                end
            end
        end
    end                                          %第一步递推
    A3 = A2 - A2(1:n,m)*A1(m,1:n)/(1+A1(m,m));  %第二步递推
    A1 = A3;
end
IA=A3;
x = A3*b;           %求解方程
```

例 11-12 叶尔索夫求逆法求线性方程组解的应用实例。用叶尔索夫求逆法求下列线性方程组的解。

$$\begin{bmatrix} 8 & 3 & 2 \\ 4 & 9 & 1 \\ 2 & 6 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解： 在 MATLAB 命令窗口中输入：

```
>> A=[8 3 2;4 9 1;2 6 10];
>> b=[1;1;1];
>> [x,IA]= Yesf(A,b)
```

输出计算结果为：

```
x =  0.0895
    0.0667
    0.0421
IA =  0.1474   -0.0316   -0.0263
    -0.0667    0.1333        0
     0.0105   -0.0737    0.1053
```

可以将求得的结果与上例比较。叶尔索夫求逆法属于消秩法求逆的一种，消秩法是一种通用的求逆矩阵的递推方法，它通过 n 步递推一定能求出可逆矩阵的逆，而叶尔索夫法是比较特殊的一种，要求矩阵各阶主子式不为零。

11.7 QR 分解法

对线性方程组 Ax = b 的系数矩阵进行 QR 分解：A=QR，则方程组可变为：

$$Rx = Q^T b$$

这是三角方程，可以用三角方程的解法求解。因此关键是怎样得到 A 的 QR 分解后中的 Q 矩阵和 R 矩阵，比较常用的方法是施密特正交化方法，用施密特正交化方法将系数矩阵变换成正交矩阵 Q 后，则 $R = Q^T A$ 。

QR 分解法的算法步骤介绍如下。

1. 将 A 施密特正交化得到 Q；
2. 令 $R = Q^T A$；
3. 求解三角方程组 $Rx = Q^T b$ 。

在 MATLAB 中编程实现的 QR 分解法函数为：qrxq
功能：QR 分解法求线性方程组 Ax=b 的解
调用格式：[x,Q,R]= qrxq (A,b)
其中，A：线性方程组的系数矩阵；
b：线性方程组中的常数向量；
x：线性方程组的解；
Q：QR 分解后的 Q 矩阵（可选的输出参数）；
R：QR 分解后的 R 矩阵（可选的输出参数）。

QR 分解法用 MATLAB 实现如下所示：

```
function [x,Q,R]=qrxq(A,b)
%QR 分解法求线性方程组 Ax=b 的解
    %线性方程组的系数矩阵：A
    %线性方程组中的常数向量：b
    %线性方程组的解：x
    % QR 分解后的 Q 矩阵：Q
    % QR 分解后的 R 矩阵：R
    N = size(A);
    n = N(1);
    B = A;                          %保存系数矩阵
    A(1:n,1)=A(1:n,1)/norm(A(1:n,1)); %将 A 的第一列正规化
    for i=2:n
      for j=1:(i-1)
          A(1:n,i)= A(1:n,i)-dot(A(1:n,i),A(1:n,j))*A(1:n,j);
          %使 A 的第 i 列与前面所有的列正交
      end
      A(1:n,i)=A(1:n,i)/norm(A(1:n,i));
      %将 A 的第 i 列正规化
    end
    Q = A;                                 %分解出来的正交矩阵
    R = transpose(Q)*B;
    x=SolveUpTriangle(R,transpose(Q)*b);  %求解方程
```

例 11-13 QR 分解法求线性方程组解的应用实例。用 QR 分解法求下列线性方程组的解。

$$\begin{bmatrix} 8 & 3 & 2 \\ 4 & 9 & 1 \\ 2 & 6 & 10 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解： 在 MATLAB 命令窗口中输入：

```
>> A=[8 3 2;4 9 1;2 6 10];
>> b=[1;1;1];
>> [x,Q,R]= qrxq(A,b)
```

输出计算结果为：

```
x =  0.0895
    0.0667
    0.0421
Q =  0.8729   -0.4811    0.0816
    0.4364    0.6949   -0.5715
    0.2182    0.5345    0.8165
R =  9.1652    7.8558    4.3644
   -0.0000    8.0178    5.0780
        0     0.0000    7.7567
```

QR 分解法是一种广受青睐的求解线性方程组的方法，从程序的代码就可以看出，它

的过程十分简捷，运算十分简单，算法复杂度是 $O(\frac{n^2}{2})$ 量级。而且即使系数矩阵是退化的情况，它也能顺利得到 Q 矩阵和 R 矩阵，可以说系数矩阵的全部信息都体现在 R 矩阵上，但它的 R 矩阵却不是通过消元法得到的，因此 QR 分解法没有消去法的各种缺点。

利用 QR 分解法可以很方便地判断线性方程组是否有解，进而求退化的线性方程组的通解，读者可以自己试试。

11.8 小结

可以说，直接法解线性方程组的思路都是最终把系数矩阵化成三角阵的形式，高斯顺序消去法的适应性较低，一旦稀疏矩阵的对角元素出现 0 的情况，它就崩溃了，因此基本上不用它来求解线性方程组。比较推荐的方法是高斯-若当消去法和 QR 分解法。高斯-若当消去法没有烦琐的行列交换，因此也常常用来求矩阵的逆，而 QR 分解法是最强大的方法，不仅适应范围广，而且速度也快，是求解线性方程组的首选方法，即使在不能确定是否有解的情况下，QR 分解法也能迅速求出方程组是否有解。

第 12 章　解线性方程组的迭代法

前面一章讨论了线性方程组的直接解法，这章介绍线性方程组的迭代解法。与直接法不同，迭代法不是通过预先规定好的有限步算术运算求得方程组的解，而是从某些初始向量出发，按一定的迭代步骤逐次算出近似解向量。

通过本章，读者不仅能掌握常见的求解线性方程组的迭代法，而且还能熟练使用 MATLAB 编程来实现这些算法。

12.1 常用迭代法

对于 n 阶线性方程组 $Ax = b$，迭代公式一般可以写成如下的形式：

$$x_{k+1} = Bx_k + r$$

其中 x_k 为第 k 次迭代的值，r 为常向量，B 称为迭代矩阵，采用不同的 B 就得到不同的迭代法。迭代法是否收敛完全取决于 B 的性态。对于无参数的迭代法，其收敛的充要条件是迭代矩阵的谱半径小于 1。

用迭代法求解线性方程组的理论在 20 世纪 50 年代就已经形成，经典的雅可比迭代法、高斯迭代法、超松弛迭代法等在那时就已经发展得十分成熟了。而后来出现的梯度法，再结合预处理技术则后来居上，取得了不少成果。

12.1.1 理查森迭代法

理查森迭代法可以说是最简单的迭代法了，它采用的迭代公式如下：

$$x_{k+1} = (I - A)x_k + b$$

其中 I 为单位矩阵。

> 在 MATLAB 中编程实现的理查森迭代法函数为：rs
>
> 功能：理查森迭代法求线性方程组 Ax=b 的解
>
> 调用格式：[x,n]=rs(A,b,x0,eps,M)
>
> 其中，A：线性方程组的系数矩阵；
>
> b：线性方程组中的常数向量；
>
> x0：迭代初始向量；
>
> eps：解的精度控制（此参数可选）；
>
> M：迭代步数控制（此参数可选）；

x：线性方程组的解；
n：求出所需精度的解实际的迭代步数。

理查森迭代法的 MATLAB 程序代码如下所示：

```
function [x,n]=rs(A,b,x0,eps,M)
%采用理查森迭代法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%迭代初始向量：x0
%解的精度控制：eps
%迭代步数控制：M
%线性方程组的解：x
%求出所需精度的解实际的迭代步数：n
if(nargin == 3)
    eps = 1.0e-6;                          %eps 表示迭代精度
      M = 10000;                           %M 表示迭代步数的限制值
elseif(nargin == 4)
      M = 10000;
end
I =eye(size(A));
n=0;
x=x0;
tol=1;
%迭代过程
while (tol>eps)
    x = (I-A)*x0+b;
    n = n + 1;                             %n 为最终求出解时的迭代步数
    tol = norm(x-x0);
    x0 = x;
    if(n>=M)
        disp('Warning:迭代次数太多，可能不收敛！');
        return;
    end
end
```

例 12-1　理查森迭代法求解线性方程组应用实例。用理查森迭代法求解线性方程组，其中取初始值为[0,0,0]。

$$\begin{bmatrix} 0.8 & -0.01 & 0.12 \\ -0.01 & 0.84 & 0.05 \\ 0.12 & 0.05 & 0.88 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解：用理查森迭代法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A=[0.8 -0.01 0.12;-0.01 0.84 0.05;0.12 0.05 0.88];
>> b=[1;1;1];
>> x0=[0;0;0];
>> [x,n]=rs(A,b,x0)
```

输出的计算结果为：

```
x =  1.1268
     1.1493
     0.9174
```

输出的迭代次数为：

```
n =  12
```

经过 12 步迭代，理查森迭代法求出了方程组的解。

对上述迭代计算结果进行验证，在 MATLAB 命令窗口中输入如下程序：

```
>> A*x
```

输出结果为：

```
ans =  1.0000
       1.0000
       1.0000
```

如此就验证了解的正确性。一般情况下，理查森迭代法难以收敛，因为要使 I-A 的谱半径小于 1，通常要求 A 是严格对角占优的矩阵。

理查森迭代法还有另外一种带迭代参数的格式：

$$x_{k+1} = (I - \omega A)x_k + \omega b$$

其中 I 为单位矩阵，ω 为迭代参数。

> 在 MATLAB 中编程实现的理查森参数迭代法函数为：crs
>
> 功能：理查森参数迭代法求线性方程组 Ax=b 的解
>
> 调用格式：[x,n]=crs(A,b,x0,w,eps,M)
>
> 其中，A：线性方程组的系数矩阵；
>
> b：线性方程组中的常数向量；
>
> x0：迭代初始向量；
>
> w：迭代参数；
>
> eps：解的精度控制（此参数可选）；
>
> M：迭代步数控制（此参数可选）；
>
> x：线性方程组的解；
>
> n：求出所需精度的解实际的迭代步数。

理查森参数迭代法的 MATLAB 程序代码如下所示：

```
function [x,n]=crs(A,b,x0,w,eps,M)
%采用理查森参数迭代法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%迭代初始向量：x0
%迭代参数：   w
```

```
%解的精度控制: eps
%迭代步数控制: M
%线性方程组的解: x
%求出所需精度的解实际的迭代步数: n
if(nargin == 4)
    eps = 1.0e-6;                              %eps 表示迭代精度
     M = 10000;                                 %M 表示迭代步数的限制值
elseif(nargin == 5)
     M = 10000;
end
I =eye(size(A));
n=0;
x=x0;
tol=1;
%迭代过程
while (tol>eps)
    x = (I-w*A)*x0+w*b;
    n = n + 1;                                  %n 为最终求出解时的迭代步数
    tol = norm(x-x0);
    x0 = x;
    if(n>=M)
        disp('Warning:迭代次数太多，可能不收敛! ');
        return;
    end
end
```

例 12-2 理查森参数迭代法求解线性方程组应用实例。用理查森参数迭代法求解下列线性方程组，其中取初始值为[0,0,0]。

$$\begin{bmatrix} 0.8 & -0.01 & 0.12 \\ -0.01 & 0.84 & 0.05 \\ 0.12 & 0.05 & 0.88 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解：用理查森参数迭代法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A=[0.8 -0.01 0.12;-0.01 0.84 0.05;0.12 0.05 0.88];
>> b=[1;1;1];
>> x0=[0;0;0];
>> [x,n]=crs(A,b,x0,1.2)
```

输出的计算结果为：

```
x =  1.1268
    1.1493
    0.9174
```

输出的迭代次数为：

```
n =  10
```

通过选取不同的迭代参数，可以减少迭代步数。

12.1.2 广义理查森迭代法

广义理查森迭代法引入了 n 个迭代参数，其迭代格式为：

$$x_{k+1}=(I-\Omega A)x_k+\Omega b$$

其中 I 为单位矩阵，Ω 是由 n 个迭代参数组成的对角阵，$\Omega=diag(\omega_1,\omega_2,\cdots,\omega_n)$。广义理查森迭代法收敛的条件是 A 对称正定，且 $2\Omega^{-1}-A$ 正定。

在 MATLAB 中编程实现的广义理查森迭代法函数为：grs

功能：理查森迭代法求线性方程组 Ax=b 的解

调用格式：[x,n]=grs(A,b,x0, W , eps, M)

其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；

x0：迭代初始向量；

W：迭代参数矩阵；

eps：解的精度控制（此参数可选）；

M：迭代步数控制（此参数可选）；

x：线性方程组的解；

n：求出所需精度的解实际的迭代步数。

广义理查森迭代法的 MATLAB 程序代码如下所示：

```
function [x,n]=grs(A,b,x0,W,eps,M)
%采用理查森迭代法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%迭代初始向量：x0
%迭代参数矩阵：W
%解的精度控制：eps
%迭代步数控制：M
%线性方程组的解：x
%求出所需精度的解实际的迭代步数：n
if(nargin == 4)
    eps = 1.0e-6;                          %eps 表示迭代精度
      M = 10000;                           %M 表示迭代步数的限制值
elseif(nargin == 5)
      M = 10000;
end
I =eye(size(A));
n=0;
x=x0;
tol=1;                                     %前后两次迭代结果误差
%迭代过程
while (tol>eps)
    x = (I-W*A)*x0+W*b;                    %迭代公式
```

```
    n = n + 1;                                    %n 为最终求出解时的迭代步数
    tol = norm(x-x0);
    x0 = x;
    if(n>=M)
        disp('Warning:迭代次数太多，可能不收敛！');
        return;
    end
end
```

例 12-3　广义理查森迭代法求解线性方程组应用实例。用广义理查森迭代法求解下列线性方程组，其中取初始值为[0,0,0]。

$$\begin{bmatrix} 0.8 & -0.01 & 0.12 \\ -0.01 & 0.84 & 0.05 \\ 0.12 & 0.05 & 0.88 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解：用广义理查森迭代法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> D=[1.1;1.2;1.21];
>> W=diag(D);
>> A=[0.8 -0.01 0.12;-0.01 0.84 0.05;0.12 0.05 0.88];
>> b=[1;1;1];
>> x0=[0;0;0];
>> [x,n]=grs(A,b,x0,W)
```

输出的计算结果为：

```
x =  1.1268
     1.1493
     0.9174
```

输出的迭代次数为：

```
n =  10
```

广义理查森迭代法采取了多个迭代参数，因此它比较容易收敛一些，此法收敛的条件是 A 对称正定，且 $2\Omega^{-1}-A$ 正定。

12.1.3　雅可比迭代法

如果系数矩阵 A 的主对角元全不为 0，取

$$B = I - D^{-1}A3$$

$$r = D^{-1}b$$

其中 D 是由 A 的主对角元素组成的对角阵。则迭代公式为：

$$x_{k+1} = (I - D^{-1}A)x_k + D^{-1}b$$

这种迭代方法称为雅可比迭代法。

在 MATLAB 中编程实现的 Jacobi 迭代法函数为：jacobi

功能：雅可比迭代法求线性方程组 Ax=b 的解

调用格式：[x,n]= jacobi (A,b,x0,eps, M)
其中，A：线性方程组的系数矩阵；
b：线性方程组中的常数向量；
x0：迭代初始向量；
eps：解的精度控制（此参数可选）；
M：迭代步数控制（此参数可选）；
x：线性方程组的解；
n：求出所需精度的解实际的迭代步数。

雅可比迭代法的 MATLAB 程序代码如下所示：

```
function [x,n]=jacobi(A,b,x0,eps,M)
%采用雅可比迭代法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%迭代初始向量：x0
%解的精度控制：eps
%迭代步数控制：M
%线性方程组的解：x
%求出所需精度的解实际的迭代步数：n
if nargin==3
    eps= 1.0e-6;
    M  = 10000;
elseif nargin ==4
    M  = 10000;
end
D=diag(diag(A));     %求 A 的对角矩阵
L=-tril(A,-1);       %求 A 的下三角阵
U=-triu(A,1);        %求 A 的上三角阵
B=D\(L+U);
f=D\b;
x=x0;
n=0;                 %迭代次数
tol=1;               %前后两次迭代结果误差
while tol>=eps
    x = B*x0+f;       %迭代公式
    n = n+1;
    tol = norm(x-x0);
    x0 = x;
    if(n>=M)
        disp('Warning: 迭代次数太多，可能不收敛！');
        return;
    end
end
```

例 12-4 雅可比迭代法求解线性方程组应用实例。用雅可比迭代法求解下列线性方程组，其中取初始值为[0,0,0]。

$$\begin{bmatrix} 0.98 & -0.05 & -0.02 \\ -0.04 & -0.9 & 0.07 \\ -0.02 & 0.09 & 0.94 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解：用雅可比迭代法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A = [ 0.98 -0.05 -0.02;-0.04 -0.9 0.07;-0.02 0.09 0.94];
>> b=[1;1;1];
>> x0=[0;0;0];
>> [x,n]=jacobi(A,b,x0)
```

输出的计算结果为：

```
x =  0.9904
   -1.0628
    1.1867
```

输出的迭代次数为：

```
n =  8
```

雅可比迭代法对任意初始向量 x_0 收敛的充要条件为 B 的谱半径小于 1。

12.1.4 高斯-赛德尔迭代法

仍然设系数矩阵 A 的主对角元全不为 0，如果对 A 作以下分解：

$$A = D - L - U$$

其中 D 的意义同雅可比迭代法，L 为下三角矩阵，U 为上三角矩阵，就得到高斯-赛德尔迭代法，它的迭代公式为：

$$x_{k+1} = (D-L)^{-1}Ux_k + (D-L)^{-1}b$$

在 MATLAB 中编程实现的高斯-赛德尔迭代法函数为：gauseidel

功能：高斯-赛德尔迭代法求线性方程组 Ax=b 的解

调用格式：[x,n]= gauseidel (A,b,x0,eps,M)

其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；

x0：迭代初始向量；

eps：解的精度控制（此参数可选）；

M：迭代步数控制（此参数可选）；

x：线性方程组的解；

n：求出所需精度的解实际的迭代步数。

高斯-赛尔德迭代法的 MATLAB 程序代码如下所示：

```
function [x,n]=gauseidel(A,b,x0,eps,M)
%采用高斯-赛德尔迭代法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
```

```
%迭代初始向量: x0
%解的精度控制: eps
%迭代步数控制: M
%线性方程组的解: x
%求出所需精度的解实际的迭代步数: n
if nargin==3
    eps= 1.0e-6;
    M  = 10000;
elseif nargin == 4
    M  = 10000;
end
D=diag(diag(A));      %求 A 的对角矩阵
L=-tril(A,-1);        %求 A 的下三角阵
U=-triu(A,1);         %求 A 的上三角阵
G=(D-L)\U;
f=(D-L)\b;
x=x0;
n=0;                  %迭代次数
tol=1;                %前后两次迭代结果误差
while tol>=eps
    x = G*x0+f;        %迭代公式
    n = n+1;
    tol = norm(x-x0);
    x0 = x;
    if(n>=M)
        disp('Warning: 迭代次数太多，可能不收敛! ');
        return;
    end
end
```

例 12-5 高斯-赛德尔迭代法求解线性方程组应用实例。用高斯-赛德尔迭代法求解下列线性方程组，其中取初始值为[0,0,0]。

$$\begin{bmatrix} 0.98 & -0.05 & -0.02 \\ -0.04 & -0.9 & 0.07 \\ -0.02 & 0.09 & 0.94 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解：用高斯-赛德尔迭代法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A = [ 0.98 -0.05 -0.02;-0.04 -0.9 0.07;-0.02 0.09 0.94];
>> b=[1;1;1];
>> x0=[0;0;0];
>> [x,n]=gauseidel(A,b,x0)
```

输出的计算结果为：

```
x =  0.9904
    -1.0628
     1.1867
```

输出的迭代次数为：

```
n =  5
```

12.1.5 超松弛迭代法

如果对 A 作以下分裂：

$$A = (D - \omega L) - ((1-\omega)D + \omega U)$$

其中 Q、L、U 的意义与高斯-赛德尔迭代法相同。

ω是一个事先选好的常数，称为松弛因子，当 $\omega>1$ 时叫超松弛法，也叫 SOR 迭代法；当 $\omega<1$ 时叫低松弛法。其迭代公式为：

$$x_{k+1} = (D-\omega L)^{-1}[(1-\omega)D + \omega U]x_k + \omega(D-\omega L)^{-1}b$$

关于超松弛迭代法的收敛性有如下结论：

若系数矩阵 A 对称正定，当 $0<\omega<2$，SOR 迭代法收敛。

在 MATLAB 中编程实现的超松弛迭代法函数为：SOR

功能：超松弛迭代法求线性方程组 Ax=b 的解

调用格式：[x,n]= SOR (A,b,x0,w,eps,M)

其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；

x0：迭代初始向量；

w：松弛因子；

eps：解的精度控制（此参数可选）；

M：迭代步数控制（此参数可选）；

x：线性方程组的解；

n：求出所需精度的解实际的迭代步数。

超松弛迭代法的 MATLAB 程序代码如下所示：

```
function [x,n]=SOR(A,b,x0,w,eps,M)
%采用超松弛迭代法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%迭代初始向量：x0
%松弛因子：w
%解的精度控制：eps
%迭代步数控制：M
%线性方程组的解：x
%求出所需精度的解实际的迭代步数：n
if nargin==4
        eps= 1.0e-6;
        M  = 10000;
    elseif nargin == 5
        M  = 10000;
    end
```

```
if(w<=0 || w>=2)
    error;
    return;
end
D=diag(diag(A));    %求 A 的对角矩阵
L=-tril(A,-1);      %求 A 的下三角阵
U=-triu(A,1);       %求 A 的上三角阵
B=inv(D-L*w)*((1-w)*D+w*U);
f=w*inv((D-L*w))*b;
x=x0;
n=0;                %迭代次数
tol=1;              %前后两次迭代结果误差
while tol>=eps
    x = B*x0+f;     %迭代公式
    n = n+1;
    tol = norm(x-x0);
    x0 = x;
    if(n>=M)
        disp('Warning: 迭代次数太多，可能不收敛！');
        return;
    end
end
```

例 12-6 超松弛迭代法求解线性方程组应用实例。用超松弛迭代法求解下列线性方程组，其中取初始值为[0,0,0]。

$$\begin{bmatrix} 0.68 & 0.01 & 0.12 \\ 0.03 & -0.54 & -0.05 \\ 0.2 & 0.08 & 0.74 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解：用超松弛迭代法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A = [0.68 0.01 0.12;0.03 -0.54 -0.05;0.2 0.08 0.74];
>> b = [1;1;1];
>> x0 = [0;0;0];
>> [x,n]=SOR(A,b,x0,1.07)
```

输出的计算结果为：

```
x =  1.2851
    -1.8924
     1.2086
```

输出的迭代次数为：

```
n =  7
```

超松弛迭代法还有一种改进形式，叫做对称逐次超松弛迭代法（SSOR），它采用的是两步迭代公式：

$$(D-\omega L)\, x_{k+1/2} = \omega(Ux_k + b) + (1-\omega)Dx_k$$
$$(D-\omega L)\, x_{k+1} = \omega(Lx_k + b) + (1-\omega)Dx_{k+1/2}$$

在 MATLAB 中编程实现的对称逐次超松弛迭代法函数为：SSOR

功能：对称逐次超松弛迭代法求线性方程组 Ax=b 的解

调用格式：[x,n]=SSOR (A,b,x0,w,eps,M)

其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；

x0：迭代初始向量；

w：松弛因子；

eps：解的精度控制（此参数可选）；

M：迭代步数控制（此参数可选）；

x：线性方程组的解；

n：求出所需精度的解实际的迭代步数。

对称逐次超松弛迭代法的 MATLAB 程序代码如下所示：

```
function [x,n]=SSOR(A,b,x0,w,eps,M)
%采用对称逐次超松弛迭代法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%迭代初始向量：x0
%松弛因子：w
%解的精度控制：eps
%迭代步数控制：M
%线性方程组的解：x
%求出所需精度的解实际的迭代步数：n
if nargin==4
    eps= 1.0e-6;
    M  = 10000;
elseif nargin ==5
    M  = 10000;
end
if(w<=0 ||  w>=2)
    error;
    return;
end
D=diag(diag(A));          %求 A 的对角矩阵
L=-tril(A,-1);            %求 A 的下三角阵
U=-triu(A,1);             %求 A 的上三角阵
B1=inv(D-L*w)*((1-w)*D+w*U);         %第一步的迭代矩阵
B2=inv(D-U*w)*((1-w)*D+w*L);         %第二步的迭代矩阵
f1=w*inv((D-L*w))*b;
f2=w*inv((D-U*w))*b;
x=x0;
n=0;                                  %迭代次数
tol=1;
%迭代过程
while tol>=eps
```

```
    x1 = B1*x0+f1;
    x  = B2*x1+f2;
    n  = n+1;
    tol = norm(x-x0);
    x0 = x;
    if(n>=M)
        disp('Warning: 迭代次数太多，可能不收敛！');
        return;
    end
end
```

例 12-7 对称逐次超松弛迭代法求解线性方程组应用实例。用对称逐次超松弛迭代法求解下列线性方程组，其中取初始值为[0,0,0]。

$$\begin{bmatrix} 0.68 & 0.01 & 0.12 \\ 0.03 & -0.54 & -0.05 \\ 0.2 & 0.08 & 0.74 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解：用对称逐次超松弛迭代法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A = [0.68 0.01 0.12;0.03 -0.54 -0.05;0.2 0.08 0.74];
>> b = [1;1;1];
>> x0 = [0;0;0];
>> [x,n]=SSOR(A,b,x0,1.07) %松弛因子为 1.07
```

输出的计算结果为：

```
x =  1.2851
    -1.8924
     1.2086
```

输出的迭代次数为：

```
n =  7
```

上面的两个例子采用的是同一个线性方程组，且采用的松弛系数都是 1.25，从求解所需的步骤来看，超松弛迭代法比对称逐次超松弛迭代法快一些。

12.1.6 雅可比超松弛迭代法

雅可比超松弛迭代法采用如下的迭代公式：

$$x_{k+1} = x_k - \omega D^{-1}(Ax_k - b)$$

ω 为迭代参数，它的大小不但影响迭代法是否收敛，而且还影响着迭代法的收敛速度，建议取 $0<\omega<2$。

> 在 MATLAB 中编程实现的雅可比超松弛迭代法函数为：JOR
> 功能：雅可比超松弛迭代法求线性方程组 Ax=b 的解
> 调用格式：[x,n]= JOR (A,b,x0,w,eps,M)
> 其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；
x0：迭代初始向量；
w：松弛因子；
eps：解的精度控制（此参数可选）；
M：迭代步数控制（此参数可选）；
x：线性方程组的解；
n：求出所需精度的解实际的迭代步数。

雅可比超松弛迭代法的 MATLAB 程序代码如下所示：

```
function [x,n]=JOR(A,b,x0,w,eps,M)
%采用雅可比超松弛迭代法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%迭代初始向量：x0
%松弛因子：w
%解的精度控制：eps
%迭代步数控制：M
%线性方程组的解：x
%求出所需精度的解实际的迭代步数：n
if nargin==4
    eps= 1.0e-6;
    M  = 10000;
elseif nargin ==5
    M  = 10000;
end
if(w<=0 || w>=2)      %收敛条件要求
    error;
    return;
end
D=diag(diag(A));     %求 A 的对角矩阵
B=w*inv(D);
%迭代过程
x=x0;
n=0;                                %迭代次数
tol=1;
%迭代过程
while tol>=eps
    x=x0-B*(A*x0-b);
    n  = n+1;
    tol  = norm(x-x0);
    x0  = x;
    if(n>=M)
        disp('Warning: 迭代次数太多，可能不收敛！');
        return;
    end
end
```

雅可比超松弛迭代法收敛的条件是 A 为对称正定矩阵，且 $2\omega D^{-1}-A$ 正定。

例 12-8 雅可比超松弛迭代法求解线性方程组应用实例。用雅可比超松弛迭代法求解下列线性方程组，其中取初始值为[0,0,0]。

$$\begin{bmatrix}0.68 & 0.01 & 0.12\\ 0.03 & -0.54 & -0.05\\ 0.2 & 0.08 & 0.74\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}=\begin{bmatrix}1\\ 1\\ 1\end{bmatrix}$$

解：用雅可比超松弛迭代法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A = [0.68 0.01 0.12;0.03 -0.54 -0.05;0.2 0.08 0.74];
>> b = [1;1;1];
>> x0 = [0;0;0];
>> [x,n]=JOR(A,b,x0,1.07)
```

输出的计算结果为：

```
x =  1.2851
   -1.8924
    1.2086
```

输出的迭代次数为：

```
n =  14
```

12.1.7 两步迭代法

两步迭代法的迭代公式如下所示：

$$(D-L)x_{k+1/2} = Ux_k + b$$
$$(D-U)x_{k+1} = Lx_{k+1/2} + b$$

其中 D、L、U 的定义和前面一样。

在 MATLAB 中编程实现的两步迭代法函数为：twostep

功能：两步迭代法求线性方程组 Ax=b 的解

调用格式：[x,n]= twostep (A,b,x0,eps, M)

其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；

x0：迭代初始向量；

eps：解的精度控制（此参数可选）；

M：迭代步数控制（此参数可选）；

x：线性方程组的解；

n：求出所需精度的解实际的迭代步数。

两步迭代法的 MATLAB 程序代码如下所示：

```
function [x,n]=twostep(A,b,x0,eps,M)
%采用两步迭代法求线性方程组 Ax=b 的解
```

```
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%迭代初始向量：x0
%解的精度控制：eps
%迭代步数控制：M
%线性方程组的解：x
%求出所需精度的解实际的迭代步数：n
if nargin==3
    eps= 1.0e-6;
    M  = 10000;
elseif nargin ==5
    M  = varargin{1};
end
D=diag(diag(A));      %求 A 的对角矩阵
L=-tril(A,-1);        %求 A 的下三角阵
U=-triu(A,1);         %求 A 的上三角阵
B1=(D-L)\U;
B2=(D-U)\L;
f1=(D-L)\b;
f2=(D-U)\b;
x=x0;
n=0;                                        %迭代次数
tol=1;
%迭代过程
while tol>=eps
    x1 = B1*x0+f1;
    x  = B2*x1+f2;
    n  = n+1;
    tol = norm(x-x0);
    x0 = x;
    if(n>=M)
        disp('Warning: 迭代次数太多，可能不收敛！ ');
        return;
    end
end
```

例 12-9　两步迭代法求解线性方程组应用实例。用两步迭代法求解下列线性方程组，其中取初始值为[0,0,0]。

$$\begin{bmatrix} 9 & 0 & 1.2 \\ 0.36 & 10 & -1.5 \\ -2.2 & 0.72 & 8 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解：用两步迭代法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A = [9 0 1.2;0.36 10 -1.5;-2.2 0.72 8];
>> b = [1;1;1];
>> x0 = [0;0;0];
>> [x,n]=JOR(A,b,x0,1.07)
```

输出的计算结果为：

```
x =  0.0925
     0.1176
     0.1398
```

输出的迭代次数为：

```
n =  5
```

从上面两个例子可以看出，与两步迭代法比，SSOR 迭代法稍快一些。

12.1.8 梯度法

梯度法相对于前面几种对系数矩阵简单分裂构造的迭代法而言，它无须考虑收敛的问题，而且求解的速度也相对较快。下面介绍梯度法的几种不同形式。能用以下三种方法求解的条件是系数矩阵对称正定。

12.1.8.1 最速下降法

最速下降法的基本思路是将求线性方程组 $Ax=b$ 的解转化为求二次泛函 $\phi(x)=xA^Tx-2b^Tx$ 的极小值问题。

通常的做法为先任意给定一个初始向量，然后确定一个搜索的方向及搜索步长，如此循环直到找到极小值。

最速下降法采取的搜索方向是当前点的负梯度方向，每次的搜索步长都使得泛函取极小值，具体的算法如下：

迭代开始

$k=0,1,2,\cdots$

$$r_k=b-Ax_k;$$

$$\alpha_k=\frac{(r_k,r_k)}{(Ar_k,r_k)};$$

$$x_{k+1}=x_k+\alpha_k r_k;$$

迭代结束



上面算法中的括号（,）代表两个向量的内积，后面算法中的括号都代表内积运算。

在 MATLAB 中编程实现的最速下降法函数为：fastdown

功能：最速下降法求线性方程组 Ax=b 的解

调用格式：[x,n]= fastdown (A,b,x0,eps)

其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；

x0：迭代初始向量；

eps：解的精度控制（此参数可选）；

x：线性方程组的解；
n：求出所需精度的解实际的迭代步数。

最速下降法的 MATLAB 程序代码如下所示：

```
    function [x,n]= fastdown(A,b,x0,eps)
    %采用最速下降法求线性方程组 Ax=b 的解
    %线性方程组的系数矩阵：A
    %线性方程组中的常数向量：b
    %迭代初始向量：x0
    %解的精度控制：eps
    %线性方程组的解：x
    %求出所需精度的解实际的迭代步数：n
if(nargin == 3)
   eps = 1.0e-6;
end
x=x0;
n=0;
tol=1;
while(tol>eps)                          %以下过程可参考算法流程
   r  = b-A*x0;
   d  = dot(r,r)/dot(A*r,r);
   x  = x0+d*r;
   tol = norm(x-x0);
   x0 = x;
   n  = n + 1;
end
```

例 12-10 最速下降法求解线性方程组应用实例。用最速下降法求解下列线性方程组，其中取初始值为[0,0,0]。

$$\begin{bmatrix} 9 & 0 & 1.2 \\ 0.36 & 10 & -1.5 \\ -2.2 & 0.72 & 8 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解：用最速下降法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A = [9 0 1.2;0.36 10 -1.5;-2.2 0.72 8];
>> b = [1;1;1];
>> x0 = [0;0;0];
>> [x,n]=fastdown(A,b,x0)
```

输出的计算结果为：

```
x =  0.0925
     0.1176
     0.1398
```

输出的迭代次数为：

```
n =  9
```

当系数矩阵为病态矩阵时，最速下降法收敛极慢，因此在实际中很少直接用它，比较常用的是由它衍生出来的下面两种方法。

12.1.8.2 共轭梯度法

共轭梯度法是从整体来寻找最佳的搜索方向。它的思路是第一步仍然取负梯度方向作为搜索方向，对以后各步，是在过当前点由负梯度向量和上一步的搜索向量组成的平面内寻找最佳的搜索方向，因此它比最速下降法快。具体的算法如下：

$$r_0 = b - Ax_0, p_0 = r_0$$

迭代开始

$k = 0,1,2,\cdots$

$$\alpha_k = \frac{(r_k, r_k)}{(Ap_k, p_k)};$$

$$x_{k+1} = x_k + \alpha_k p_k;$$

$$r_{k+1} = r_k - \alpha_k A p_k;$$

$$\beta_k = \frac{(r_{k+1}, r_{k+1})}{(r_k, r_k)};$$

$$p_{k+1} = r_{k+1} + \beta_k p_k;$$

迭代结束

在 MATLAB 中编程实现的共轭梯度法函数为：conjgrad

功能：共轭梯度法求线性方程组 Ax=b 的解

调用格式：[x,n]= conjgrad (A,b,x0)

其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；

x0：迭代初始向量；

x：线性方程组的解；

n：求出所需精度的解实际的迭代步数。

共轭梯度法的 MATLAB 程序代码如下所示：

```
function [x,n]= conjgrad (A,b,x0)
%采用共轭梯度法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%迭代初始向量：x0
%线性方程组的解：x
%求出所需精度的解实际的迭代步数：n
r1  = b-A*x0;
p  = r1;
n  = 0;
for i=1:rank(A)                    %以下过程可参考算法流程
    if(dot(p,A*p) < 1.0e-50)       %循环结束条件
        break;
```

```
    end
    alpha  = dot(r1,r1)/dot(p,A*p);
    x  = x0+ alpha*p;
    r2 = r1- alpha*A*p;
    if(r2 < 1.0e-50)              %循环结束条件
         break;
    end
    belta  = dot(r2,r2)/dot(r1,r1);
    p  = r2+belta*p;
    n  = n + 1;
end
```

例 12-11 共轭梯度法求解线性方程组应用实例。用共轭梯度法求解下列线性方程组，其中取初始值为[0,0,0]。

$$\begin{bmatrix} 9 & 0 & 1.2 \\ 0.36 & 10 & -1.5 \\ -2.2 & 0.72 & 8 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解：用共轭梯度法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A = [9 0 1.2;0.36 10 -1.5;-2.2 0.72 8];
>> b = [1;1;1];
>> x0 = [0;0;0];
>> [x,n]= conjgrad(A,b,x0)
```

输出的计算结果为：

```
x =  0.0906
     0.1186
     0.1445
```

输出的迭代次数为：

```
n =  3
```

共轭梯度法理论上只要迭代 n 步，就能得出方程组的解，但是由于存在计算误差，即分数向小数转化时存在舍入误差，很难保证在第 n 步时得到准确解。因此在实际使用中，当第 n 步还没得到要求精度的解时，继续迭代下去，程序只需稍微修改循环的结束条件即可：

```
......
tol=1;
while(tol>eps)                    %以下过程可参考算法流程
    ......
    tol = norm(x-x0);         %前后两次迭代解的差值
    ......
```

共轭梯度法求解的速度非常快，比最速下降法提高了好几个数量级。

12.1.8.3　预处理共轭梯度法

当 $Ax=b$ 为病态方程组时，共轭梯度法会收敛得很慢。预处理技术是在用共轭梯度法

求解之前对系数矩阵做一些变换再迭代求解。

设$M=SS^T$，在方程组两边都乘以一个非奇异矩阵S^{-1}，令

$$y=S^Tx$$
$$c=S^{-1}b$$
$$B=S^{-1}AS^{-T}$$

则原方程组可写成：

$$Bu=c$$

再对上面的线性方程组用共轭梯度法求解。预处理共轭梯度法的整个算法过程如下：

$$r_0=b-Ax_0, z_0=M^{-1}r_0, p_0=z_0$$

迭代开始

$k=0,1,2,\cdots$

$$\alpha_k=\frac{(r_k,z_k)}{(Ap_k,p_k)};$$
$$x_{k+1}=x_k+\alpha_k p_k;$$
$$r_{k+1}=r_k-\alpha_k Ap_k;$$
$$z_{k+1}=M^{-1}r_{k+1};$$
$$\beta_k=\frac{(r_{k+1},z_{k+1})}{(r_k,z_k)};$$
$$p_{k+1}=z_{k+1}+\beta_k p_k;$$

迭代结束

在 MATLAB 中编程实现的预处理共轭梯度法函数为：preconjgrad

功能：预处理共轭梯度法求线性方程组 Ax=b 的解

调用格式：[x,n]= preconjgrad (A,b,x0,M,eps)

其中，A：线性方程组的系数矩阵；
- b：线性方程组中的常数向量；
- x0：迭代初始向量；
- M：预处理矩阵；
- eps：精度控制（可选参数）；
- x：线性方程组的解；
- n：求出所需精度的解实际的迭代步数。

预处理共轭梯度法的 MATLAB 程序代码如下所示：

```
function [x,n]= preconjgrad (A,b,x0,M,eps)
%采用共轭梯度法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%迭代初始向量：x0
%预处理矩阵： M
%精度控制：    eps
```

```
%线性方程组的解: x
%求出所需精度的解实际的迭代步数: n
function [x,n]=preconjgrad(A,b,x0,M,eps)
if nargin == 4
    eps = 1.0e-6;
end
r1 = b-A*x0;
iM = inv(M);
z1 = iM*r1;
p  = z1;
n  = 0;
tol= 1;
while tol>=eps
    alpha  = dot(r1,z1)/dot(p,A*p);
    x  = x0 + alpha*p;
    r2 = r1 - alpha*A*p;
    z2 = iM*r2;
    belta  = dot(r2,z2)/dot(r1,z1);
    p  = z2+belta*p;
    n  = n + 1;
    tol = norm(x-x0);
    x0 = x;                    %更新迭代值
    r1 = r2;
    z1 = z2;
end
```

预处理对收敛性的改善主要是通过矩阵 M 实现的，因此 M 的选择就比较重要了。一般情况下，M 有以下三种选择方式：

- 选取 M 为 A 的对角元素组成的对角阵，这种方法简单，但是收敛效果不理想；
- 将 A 作不完全乔列斯基分解为 $A = LL^T - R$，取 $M = LL^T$，此种方法是比较常用的 ICCG 法；
- 取 M 为对称超松弛迭代法的预处理阵 $M = SS^T$，其中

$$S = [\omega(2-\omega)]^{-1/2}(D-\omega L)D^{-1/2}$$

经过这样的预处理，能够显著改善收敛速度，特别是当 $\omega = 1$ 时，预处理会有很好的效果。

例 12-12 预处理共轭梯度法求解线性方程组应用实例。用预处理共轭梯度法求解下列线性方程组，其中取初始值为[0,0,0]。

$$\begin{bmatrix} 90 & 0 & 1.2 \\ 0.36 & 100 & -1.5 \\ -2.2 & 0.72 & 80 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解：用预处理共轭梯度法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A = [90 0 1.2;0.36 100 -1.5;-2.2 0.72 80];
>> b = [1;1;1];
>> x0 = [0;0;0];
```

```
>> D=diag(diag(A));
>> L=(-tril(A)+2*D)*sqrt(D);
>> M=L*L';
>> [x,n]= preconjgrad(A,b,x0,M)
```

输出的计算结果为:

```
x =  0.0109
    0.0102
    0.0127
```

输出的迭代次数为:

```
n = 5
```

如果预处理矩阵 M 取得适当的话，能很好地改善系数矩阵的条件数，因此在处理病态矩阵方面，预处理技术的效果是十分明显的。

12.1.9 块迭代法

当系数矩阵的阶数比较大时，可以先将它分块，再进行迭代求解。

$$A=\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mm} \end{bmatrix}$$

其中 $A_{11},A_{22},\cdots A_{mm}$ 为方阵。

再将系数矩阵写成如下的形式:

$$A=D_B-L_B-U_B$$

其中:

$$D_B=diag(A_{11},A_{12},\cdots,A_{1m})$$

$$L_B=-\begin{bmatrix} 0 & & & & \\ A_{21} & 0 & & & \\ A_{31} & A_{32} & 0 & & \\ \vdots & \vdots & \ddots & \ddots & \\ A_{m1} & A_{m2} & A_{m3} & A_{m,m-1} & 0 \end{bmatrix}$$

$$U_B=-\begin{bmatrix} 0 & A_{12} & A_{13} & \cdots & A_{1m} \\ & 0 & A_{23} & \cdots & A_{2m} \\ & & 0 & \ddots & \vdots \\ & & & \ddots & A_{m-1,m} \\ & & & & 0 \end{bmatrix}$$

当然也要将 x 和 b 作相应的分块:

$$x=\begin{bmatrix}x_1\\x_2\\\vdots\\x_m\end{bmatrix},b=\begin{bmatrix}b_1\\b_2\\\vdots\\b_m\end{bmatrix}$$

下面是几种常用的块迭代法，当然分块后的线性方程组可以使用各种迭代法进行求解。

12.1.9.1 块雅克比迭代法

块雅克比迭代法的块迭代公式如下所示：

$$x_{k+1}=D_B^{-1}(L_B+U_B)x_k+D_B^{-1}b$$

写成分量形式如下所示：

$$A_{ii}x_i^{(k+1)}=b_i-\sum_{\substack{j=1\\j\neq i}}^{m}A_{ij}x_j^k$$

在 MATLAB 中编程实现的块雅克比迭代法函数为：BJ

功能：块雅克比迭代法求线性方程组 Ax=b 的解

调用格式：[x,N]= BJ (A,b,x0,eps, d,eps,M)

其中，A：线性方程组的系数矩阵；
- b：线性方程组中的常数向量；
- x0：迭代初始向量；
- d：系数矩阵沿对角线的分块向量（列向量）；
- eps：解的精度控制（此参数可选）；
- M：迭代步数控制（此参数可选）；
- x：线性方程组的解；
- N：求出所需精度的解实际的迭代步数。



上面参数中 d 的意义如下，假如 A 为 5 阶的方阵，则 $d=[1\ \ 2\ \ 2]^T$ 表示将 A 分成 9 块，其中对角线上的 A_{11} 为 1 阶方阵，A_{22} 为 2 阶方阵，A_{33} 为 2 阶方阵。

块雅克比迭代法的 MATLAB 程序代码如下所示：

```
function [x,N]= BJ(A,b,x0,d,eps,M)
%采用块雅克比迭代法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%迭代初始向量：x0
%系数矩阵沿对角线的分块向量(列向量)：d
%解的精度控制：eps
%迭代步数控制：M
%线性方程组的解：x
%求出所需精度的解实际的迭代步数：N
```

```
if nargin==4
    eps= 1.0e-6;
    M  = 10000;
elseif nargin<4
    error
    return
elseif nargin ==5
    M  = 10000;                 %参数的默认值
end
NS = size(A);
n = NS(1,1);
if(sum(d)  ~= n)
    disp('分块错误！');
    return;
end
bnum = length(d);
bs = ones(bnum,1);
for i=1:(bnum-1)
    bs(i+1,1)=sum(d(1:i))+1;
    %获得对角线上每个分块矩阵元素索引的起始值
end
DB = zeros(n,n);
for i=1:bnum
    endb = bs(i,1)+d(i,1)-1;
    DB(bs(i,1):endb,bs(i,1):endb)=A(bs(i,1):endb,bs(i,1):endb);
    %求 A 的对角分块矩阵
end
for i=1:bnum
    endb = bs(i,1)+d(i,1)-1;
    invDB(bs(i,1):endb,bs(i,1):endb)=inv(DB(bs(i,1):endb,bs(i,1):endb));
    %求 A 的对角分块矩阵的逆矩阵
end
N = 0;
tol = 1;
while tol>=eps
    x = invDB*(DB-A)*x0+invDB*b;     %由于 LB+DB=DB-A
    N = N+1;                            %迭代步数
    tol = norm(x-x0);                  %前后两步迭代结果的误差
    x0 = x;
    if(N>=M)
        disp('Warning: 迭代次数太多，可能不收敛！');
        return;
    end
end
```

例 12-13 块雅克比迭代法求解线性方程组应用实例。用块雅克比迭代法求解下列线性方程组，其中取初始值为[0,0,0,0,0]。

$$\begin{bmatrix} 2.6934 & 0.6901 & 0.3997 & 0.6010 & 0.4390 \\ 0.6901 & 2.8784 & 0.8799 & 0.5978 & 0.4514 \\ 0.3997 & 0.8799 & 3.3216 & 0.4673 & 0.8282 \\ 0.6010 & 0.5978 & 0.4673 & 2.8412 & 0.5511 \\ 0.4390 & 0.4514 & 0.8282 & 0.5511 & 2.8704 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

解：用块雅克比迭代法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A=[2.6934 0.6901 0.3997 0.6010 0.4390;
       0.6901 2.8784 0.8799 0.5978 0.4514;
       0.3997 0.8799 3.3216 0.4673 0.8282;
       0.6010 0.5978 0.4673 2.8412 0.5511;
       0.4390  0.4514 0.8282  0.5511 2.8704];
>> d=[1;2;2];
>> [x,n]=BJ(A,b,[0 0 0 0 0]',d)
x =  0.2260
     0.1735
     0.1482
     0.2036
     0.2047
n =  23
>> d=[4;1];
>> [x,n]=BJ(A,b,[0 0 0 0 0]',d)
x =  0.2260
     0.1735
     0.1482
     0.2036
     0.2047
n =  13
>> [x,n]=jacobi(A,b,[0 0 0 0 0]')
x =  0.2260
     0.1735
     0.1482
     0.2036
     0.2047
n =  66
```

从上面的例子可以看出，分块的方式决定了块雅克比迭代法收敛的快慢。最快经过 13 步迭代就得到了解，而最慢的则花了 66 步（不分块的雅可比迭代法是块雅克比迭代法的退化情况），根据上例 d 的分块情况可以得出一个规律，即分得越散，收敛速度越慢。因此，块不宜分得太多。

12.1.9.2 块高斯-赛德尔迭代法

块高斯-赛德尔迭代法的块迭代公式如下所示：

$$x_{k+1} = (D_B - L_B)^{-1} U_B x_k + (D_B - L_B)^{-1} b$$

写成分量形式如下所示：

$$A_{ii}x_i^{(k+1)} = b_i - \sum_{j=1}^{i-1} A_{ij}x_j^{k+1} - \sum_{j=i+1}^{m} A_{ij}x_j^k$$

在 MATLAB 中编程实现的块高斯-赛德尔迭代法函数为：BGS

功能：块高斯-赛德尔迭代法求线性方程组 Ax=b 的解

调用格式：[x,N]= BGS (A,b, x0,eps, d,eps,M)

其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；

x0：迭代初始向量；

d：系数矩阵沿对角线的分块向量(列向量)；

eps：解的精度控制（此参数可选）；

M：迭代步数控制（此参数可选）；

x：线性方程组的解；

N：求出所需精度的解实际的迭代步数。

块高斯-赛德尔迭代法的 MATLAB 程序代码如下所示：

```
function [x,N]= BGS(A,b,x0,d,eps,M)
%采用块高斯-赛德尔迭代法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%迭代初始向量：x0
%系数矩阵沿对角线的分块向量(列向量)：d
%解的精度控制：eps
%迭代步数控制：M
%线性方程组的解：x
%求出所需精度的解实际的迭代步数：N
if nargin==4
    eps= 1.0e-6;
    M  = 10000;
elseif nargin<4
    error
    return
elseif nargin ==5
    M  = 10000;
end
NS = size(A);
n = NS(1,1);
bnum = length(d);
bs = ones(bnum,1);
for i=1:(bnum-1)
    bs(i+1,1)=sum(d(1:i))+1;
    %获得对角线上每个分块矩阵元素索引的起始值
end
DB = zeros(n,n);
for i=1:bnum
```

```
        endb = bs(i,1)+d(i,1)-1;
        DB(bs(i,1):endb,bs(i,1):endb)=A(bs(i,1):endb,bs(i,1):endb);
        %求 A 的对角分块矩阵
    end
    LB = -tril(A-DB);        %求 A 的下三角分块阵
    UB = -triu(A-DB);        %求 A 的上三角分块阵
    N = 0;
    tol = 1;
    while tol>=eps
        invDL = inv(DB-LB);
        x = invDL*UB*x0+invDL*b;     %块迭代公式
        N = N+1;
        tol = norm(x-x0);
        x0 = x;
        if(N>=M)
            disp('Warning: 迭代次数太多，可能不收敛！');
            return;
        end
    end
```

例 12-14 块高斯-赛德尔迭代法求解线性方程组应用实例。用块高斯-赛德尔迭代法求解下列线性方程组，其中取初始值为[0,0,0,0,0]。

$$\begin{bmatrix} 2.6934 & 0.6901 & 0.3997 & 0.6010 & 0.4390 \\ 0.6901 & 2.8784 & 0.8799 & 0.5978 & 0.4514 \\ 0.3997 & 0.8799 & 3.3216 & 0.4673 & 0.8282 \\ 0.6010 & 0.5978 & 0.4673 & 2.8412 & 0.5511 \\ 0.4390 & 0.4514 & 0.8282 & 0.5511 & 2.8704 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

解：用块高斯-赛德尔迭代法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A=[2.6934 0.6901 0.3997 0.6010 0.4390;
      0.6901 2.8784 0.8799 0.5978 0.4514;
      0.3997 0.8799 3.3216 0.4673 0.8282;
      0.6010 0.5978 0.4673 2.8412 0.5511;
      0.4390  0.4514 0.8282  0.5511 2.8704];
>> d=[1;2;2];
>> [x,n]=BGS(A,b,[0  0  0  0  0]',d)
x =  0.2260
     0.1735
     0.1482
     0.2036
     0.2047
n =  8
```

在上面的例子中，只需要经过 8 步迭代就得到了解，比块雅可比迭代法更进了一步。但是从计算量来说，块雅可比迭代法只需要求一个对角矩阵的逆即可，而块高斯-赛德尔迭代法要计算一个下三角矩阵的逆，显然比求对角矩阵的逆复杂。

为了避免求逆，可以采取下面的迭代公式：

$$y_k = U_B x_k + b$$

$$(D_B - L_B)x_{k+1} = y_k$$

其中第二个方程可以用第 15 章中的求解下三角系数矩阵线性方程组的函数求解，读者可以自己把程序写出来。

12.1.9.3 块逐次超松弛迭代法

块逐次超松弛迭代法的块迭代公式如下所示：

$$x_{k+1} = (D_B - \omega L_B)^{-1}((1-\omega)D_B + \omega U_B)x_k + \omega(D_B - \omega L_B)^{-1}b$$

写成分量形式如下所示：

$$A_{ii}x_i^{(k+1)} = (1-\omega)A_{ii}x_i^k + \omega(b_i - \sum_{\substack{j=1 \\ j\neq i}}^{i-1} A_{ij}x_j^{k+1} - \sum_{\substack{j=i+1 \\ j\neq i}}^{m} A_{ij}x_j^k)$$

在 MATLAB 中编程实现的块逐次超松弛迭代法函数为：BSOR

功能：块逐次超松弛迭代法求线性方程组 Ax=b 的解

调用格式：[x,N]= BSOR (A,b, x0,eps, d,w,eps,M)

其中，A：线性方程组的系数矩阵；

b：线性方程组中的常数向量；

x0：迭代初始向量；

d：系数矩阵沿对角线的分块向量(列向量)；

w：松弛因子；

eps：解的精度控制（此参数可选）；

M：迭代步数控制（此参数可选）；

x：线性方程组的解；

N：求出所需精度的解实际的迭代步数。

块逐次超松弛迭代法的 MATLAB 程序代码如下所示：

```
function [x,N]= BSOR(A,b,x0,d,w,eps,M)
%采用块逐次超松弛迭代法求线性方程组 Ax=b 的解
%线性方程组的系数矩阵：A
%线性方程组中的常数向量：b
%迭代初始向量：x0
%系数矩阵沿对角线的分块向量(列向量)：d
%松弛因子：w
%解的精度控制：eps
%迭代步数控制：M
%线性方程组的解：x
%求出所需精度的解实际的迭代步数：N
if nargin==5
    eps= 1.0e-6;
    M  = 10000;
elseif nargin<5
    error
```

```
    return
elseif nargin ==6
    M  = 10000;             %参数默认值
end
NS = size(A);
n = NS(1,1);
bnum = length(d);
bs = ones(bnum,1);
for i=1:(bnum-1)
    bs(i+1,1)=sum(d(1:i))+1;
    %获得对角线上每个分块矩阵元素索引的起始值
end
DB = zeros(n,n);
for i=1:bnum
    endb = bs(i,1)+d(i,1)-1;
    DB(bs(i,1):endb,bs(i,1):endb)=A(bs(i,1):endb,bs(i,1):endb);
    %求 A 的对角矩阵
end
LB = -tril(A-DB);       %求 A 的下三角阵
UB = -triu(A-DB);       %求 A 的上三角阵
N = 0;
tol = 1;
iw = 1-w;
while tol>=eps
    invDL = inv(DB-w*LB);
    x = invDL*(iw*DB+w*UB)*x0+w*invDL*b;    %块迭代公式
    N = N+1;
    tol = norm(x-x0);
    x0 = x;
    if(N>=M)
        disp('Warning: 迭代次数太多，可能不收敛！');
        return;
    end
end
```

例 12-15 块逐次超松弛迭代法求线性方程组应用实例。用块逐次超松弛迭代法求解下列线性方程组，其中取初始值为[0,0,0,0,0]。

$$\begin{bmatrix} 2.6934 & 0.6901 & 0.3997 & 0.6010 & 0.4390 \\ 0.6901 & 2.8784 & 0.8799 & 0.5978 & 0.4514 \\ 0.3997 & 0.8799 & 3.3216 & 0.4673 & 0.8282 \\ 0.6010 & 0.5978 & 0.4673 & 2.8412 & 0.5511 \\ 0.4390 & 0.4514 & 0.8282 & 0.5511 & 2.8704 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

解：用块逐次超松弛迭代法求解，在 MATLAB 命令窗口中输入求解程序：

```
>> A=[2.6934 0.6901 0.3997 0.6010 0.4390;
      0.6901 2.8784 0.8799 0.5978 0.4514;
      0.3997 0.8799 3.3216 0.4673 0.8282;
```

```
        0.6010 0.5978 0.4673 2.8412 0.5511;
        0.4390  0.4514 0.8282  0.5511 2.8704];
>> d=[1;2;2];
>> [x,n]=BSOR(A,b,[0 0 0 0 0]',d,1.1) %松弛因子取1.1
x =  0.2260
    0.1735
    0.1482
    0.2036
    0.2047
n =  8
```

为了避免求逆运算，同样可以采取块高斯-赛德尔迭代法的改进做法。

12.2 小结

迭代法显著的特点是算法简单，因此程序实现起来也比较简单，它是解大型线性方程组的有效方法。

古典的迭代法思路很简单，用到的知识不多，但是适用范围不广；由于存储技术的进步，现代的迭代法则注重算法的适用性和收敛性，追求适用范围广，收敛性好，收敛速度快，预处理技术就是其中的一个方向，将预处理技术与一些特殊的迭代法相结合往往能收到很好的效果。

第 13 章　随机数生成

不管是在计算机编程，还是网站设计、分析实际问题，随机数都有广泛的应用。本章首先讨论生成 0～1 之间均匀分布随机数的一些算法，进而给出了由 0～1 之间均匀分布生成指数分布、正态分布、泊松分布、二项分布等常用分布的一般算法。

通过本章，读者不仅能掌握常见的随机数生成算法，而且还能熟练使用 MATLAB 编程来实现这些算法。

13.1 平方取中法

把一个 $2k$ 位 t 进制的数平方，再把平方之后的数去头截尾取中间 $2k$ 位数作为一个新的随机数，重复上述过程，则得到一列随机数，递推公式为：

$$\begin{cases} x_n = \left(\dfrac{x_{n-1}^2}{10^k}\right)(\text{mod} \quad 10^{2k}) \\ r_n = 10^{-2k} x_n \qquad\qquad n=1,2,\cdots \end{cases}$$

数列 $r_1, r_2, \cdots$ 为 $[0,1]$ 上均匀分布的随机数列。

> 在 MATLAB 中编程实现的平方取中法的函数为：PFQZ
>
> 功能：用平方取中法产生随机数列
>
> 调用格式：r = PFQZ(k,x0,n)
>
> 其中，k：随机数种子位数的一半；
>
> x0：随机数种子；
>
> n：产生的随机数个数；
>
> r：产生的随机数序列。

平方取中法产生随机数列的 MATLAB 程序代码如下所示：

```
function r = PFQZ(k,x0,n)
    %随机数种子位数的一半：k
    %随机数种子：x0
    %产生的随机数个数：n
    %产生的随机数序列：r
    format long;
    r = zeros(n,1);
    x = zeros(n,1);
    x(1) = x0;
```

```
    r(1) = x(1)/(100^k);
for i=2:n
    x(i) = mod(x(i-1)^2 /(10^k),100^k);   %平方取中
    r(i) = x(i)/(100^k);                   %转换到区间[0,1]
end
format short;
```

例 13-1 平方取中法产生随机数列应用实例。用平方取中法产生 10 个随机数。

解： 在 MATLAB 中输入下列命令：

```
>> r = PFQZ(2,3456,10)
r =  0.3456
     0.9439
     0.1015
     0.0306
     0.0935
     0.8738
     0.3593
     0.9077
     0.3892
     0.1449
```

可以用如下的程序检验产生的随机数是否是[0,1]上均匀分布的随机数列：

```
function numDist = test(k,x0,N)
numDist = zeros(10,1);
x = PFQZ(k,x0,N);
for i=1:N
    y = x(i)*10;
    j = floor(y);
    numDist(j+1) = numDist(j+1) + 1;
end
t=1:10;
bar(t,numDist);
```

在 MATLAB 中输入下列命令：

```
>> num = test(2,5466,5000)
num =  492
       516
       480
       520
       503
       486
       472
       485
       506
       540
```

而且可以用如图 13-1 所示的直方图明显看出来：

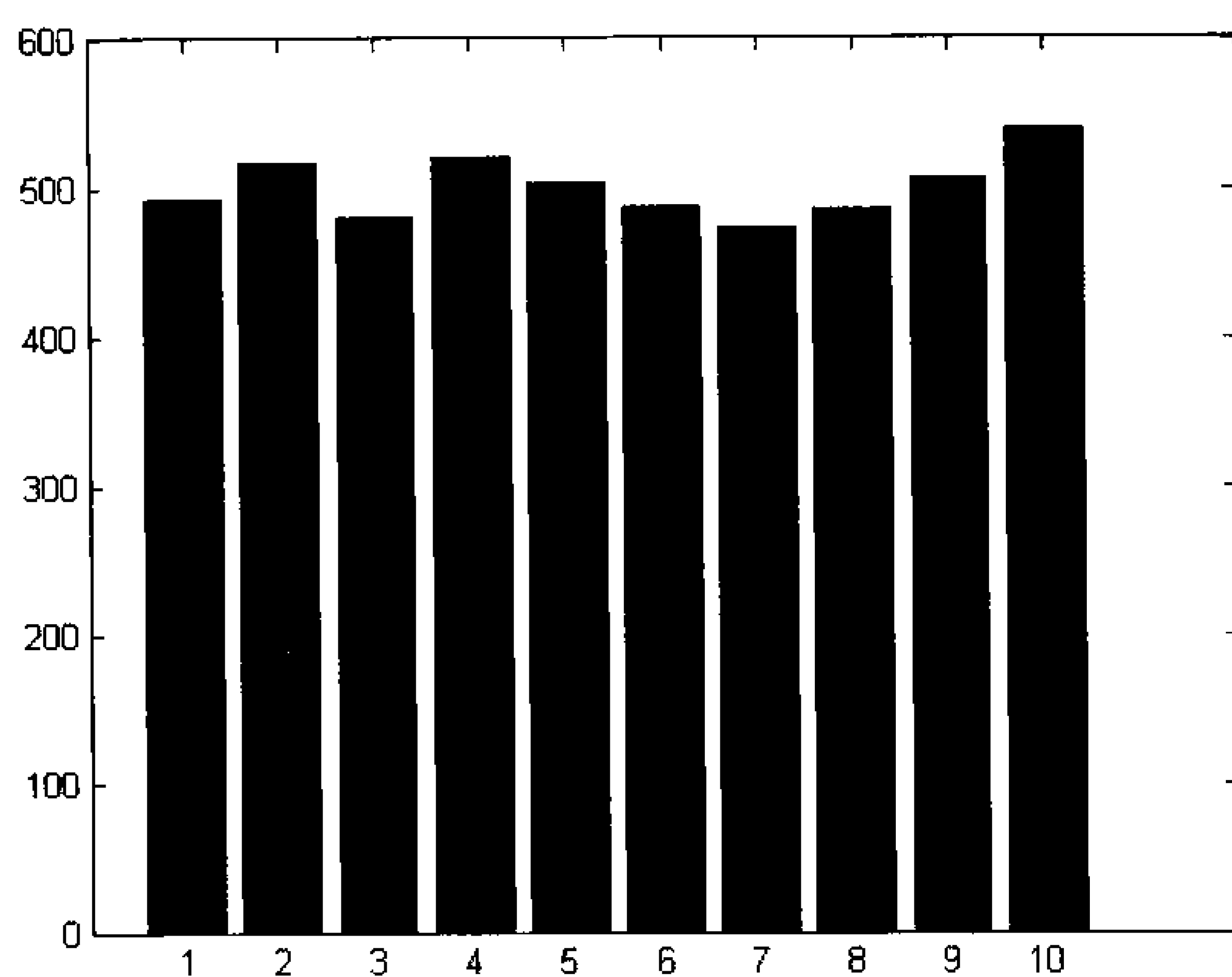


图 13-1 用平方取中法产生的随机数的分布图

在图 13-1 所示的直方图中，横坐标表示随机数所落在的区间，例如 1 表示区间[0.0,0.1]，6 表示区间[0.5,0.6]，依此类推；纵坐标表示对应区间的随机数的个数。从直方图上可以看出，5000 个随机数，基本上平均分布在区间[0,1]上。

13.2 线性同余法

线性同余法简称为 LCG 法，它利用数论中的同余运算原理产生随机数，它又可分为混合同余法、乘同余法等。线性同余法的一般公式为：

$$\begin{cases} x_n = (ax_{n-1} + c) \bmod M \\ r_n = x_n / M \end{cases}$$

其中 x_0 为初值，M 为模，第二个公式的作用是将产生的随机数变换到区间[0,1]上。当 $c=0$ 时为乘同余法，当 $c>0$ 时为混合同余法。因此，当提供初始值（也可叫种子）x_0 后，就可以通过递推公式产生一随机数列 $x_0, x_1, \cdots x_n, \cdots$。

13.2.1 混合同余法

混合同余法的递推公式为：

$$\begin{cases} x_n = (ax_{n-1} + c) \bmod M, c > 0 \\ r_n = x_n / M \end{cases}$$

通过取不同的 M、a、c、x_0 可得到不同的随机数列，但是产生的随机数列是周期随机数列，而且存在常数，使得

$$x_{n+T} = x_n、(n = 0,1,2,\cdots)$$

如果 $T = M$，则称此随机数列为满周期随机数列。

一般情况下，参数的选取可遵循下述原则：

- $M = 2^L$；
- $a = 4\alpha + 1, \alpha$ 为任意整数；

- $c = 2\beta + 1, \beta$ 为任意整数；
- x_0 为任意非负整数。

常用的两种混合同余法公式如下所示：

- $$\begin{cases} x_n = (314159269 x_{n-1} + 453806245) \bmod 2^{31} \\ r_n = x_n / 2^{31} \end{cases}$$
- $$\begin{cases} x_n = (5^{15} x_{n-1} + 1) \bmod 2^{35} \\ r_n = x_n / 2^{35} \end{cases}$$

在 MATLAB 中编程实现的混合同余法的函数为：MixMOD

功能：用混合同余法产生随机数列

调用格式：r = MixMOD(x0,n,type)

其中，x0：随机数种子；

n：产生的随机数个数；

type：采用的混合同余法的公式类型；

r：产生的随机数序列。

混合同余法的 MATLAB 程序代码如下：

```
function r = MixMOD(x0,n,type)
%随机数种子：x0
%产生的随机数个数：n
%采用的混合同余法的公式类型：type
%产生的随机数序列：r
format long;
M1 = power(2,31);
M2 = power(2,35);
a1 = 314159269;
a2 = power(5,15);
c1 = 453806245;
c2 = 1;
r = zeros(n,1);
x = zeros(n+1,1);
x(1) = x0;
if type == 1      %L 为 31 的混合同余法
    for i=2:n+1
        y = a1*x(i-1)+c1;
        x(i) = mod(y, M1);
        r(i-1) = x(i)/M1;
    end
else                %L 为 35 的混合同余法
    for i=2:n+1
        y = a2*x(i-1)+c2;
        x(i) = mod(y, M2);
        r(i-1) = x(i)/M2;
    end
```

```
end
format short;
```

例 13-2 混合同余法产生随机数列应用实例。用混合同余法产生 10 个随机数。

解： 在 MATLAB 中输入下列命令：

```
>> r = MixMOD(0,10,1)
r =  0.2113
     0.0102
     0.4809
     0.6091
     0.3940
     0.7675
     0.6389
     0.7911
     0.8903
     0.6004
>> r = MixMOD(1,10,2)
r =  0.8882
     0.0258
     0.8719
     0.3734
     0.1676
     0.8227
     0.3805
     0.9149
     0.2939
     0.5816
```

可以用如下的程序检验产生的随机数是否是[0,1]上均匀分布的随机数列：

```
function numDist = test(x0,N)
numDist = zeros(10,1);
x = MixMOD(x0,N,2);
for i=1:N
    y = x(i)*10;
    j = floor(y);
    numDist(j+1) = numDist(j+1) + 1;
end
t=1:10;
bar(t,numDist);
```

在 MATLAB 中输入下列命令：

```
>> num = test(3345,100000)
num =  9975
      10017
      10015
       9988
      10027
       9987
       9998
```

```
    10004
     9992
     9997
```

而且可以从如图 13-2 所示的直方图中明显看出来：

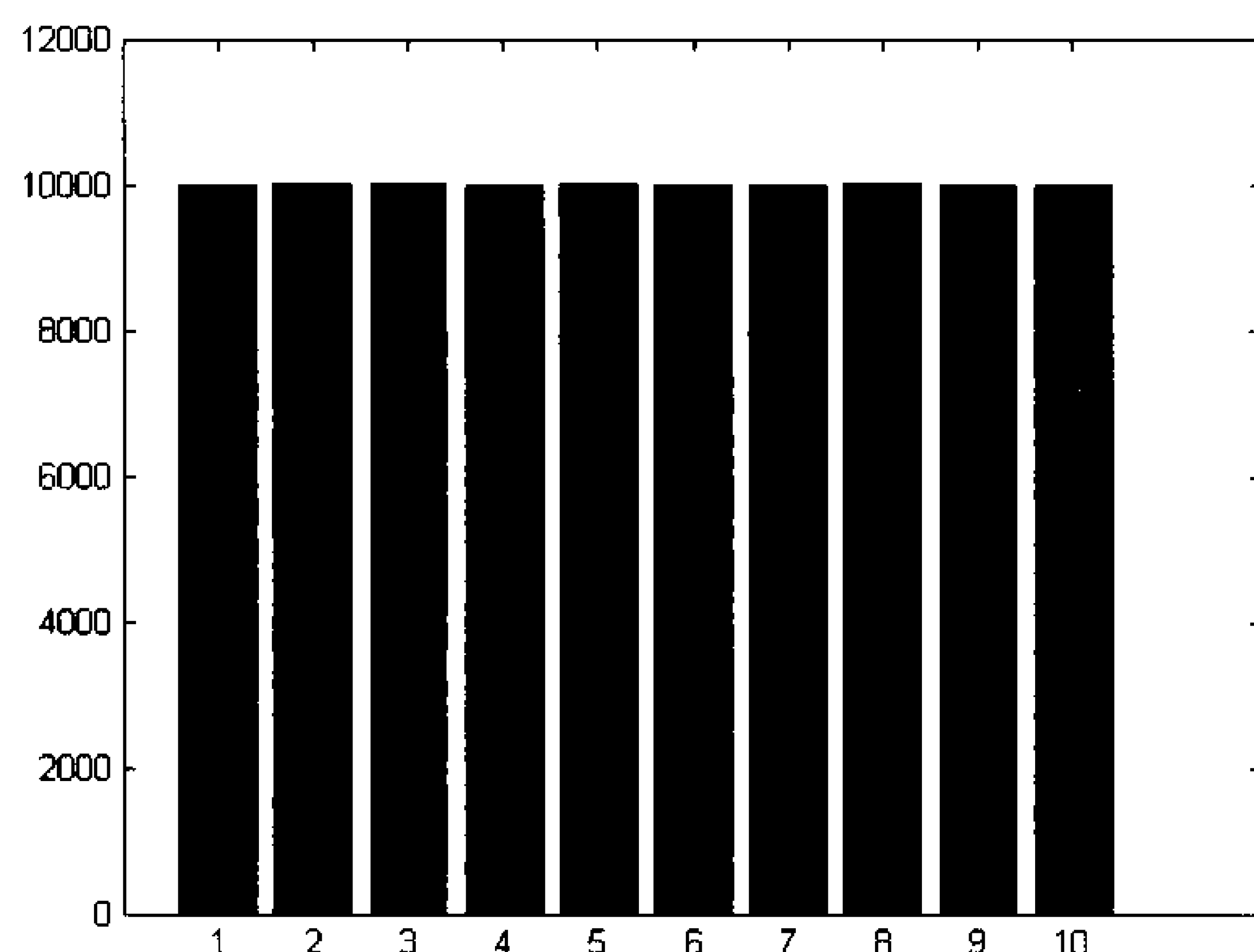


图 13-2　用混合同余法产生的随机数的分布图

13.2.2　乘同余法

乘同余法的递推公式为：

$$\begin{cases} x_n = a x_{n-1} \mod M \\ r_n = x_n / M \end{cases}$$

一般情况下，参数的选取有两种方法。

- 取 $M = 2^L$，L 为任意正整数；
  $a = 8\alpha \pm 5, \alpha$ 为任意整数；
  $x_0 = 4\beta + 1, \beta$ 为任意整数。

> 在 MATLAB 中编程实现的乘同余法 1 的函数为：MulMOD1
> 功能：用乘同余法 1 产生随机数列
> 调用格式：r = MulMOD1(L,alpha,beta,n)
> 其中，L：模的幂参数；
> 　　alpha：乘数的参数；
> 　　beta：随机数种子的参数；
> 　　n：产生的随机数个数；
> 　　r：产生的随机数序列。

乘同余法 1 的 MATLAB 程序代码如下所示：

```
function r = MulMOD1(L,alpha,beta,n)
%模的幂参数：L
%乘数的参数：alpha
%随机数种子的参数：beta
```

```
%产生的随机数个数：n
%产生的随机数序列：r
r = zeros(n,1);
x = zeros(n,1);
M = power(2,L);
a = 8*alpha + 5;
x(1) = 4*beta + 1;
r(1) = x(1)/M;
for i=2:n
    y = a*x(i-1);
    x(i) = mod(y,M);    %乘同余法
    r(i) = x(i)/M;      %转换到区间[0,1]
end
```

例 13-3　乘同余法 1 产生随机数列应用实例。用乘同余法 1 产生 10 个随机数。

解：在 MATLAB 中输入下列命令：

```
>> r = MulMOD1(13,64,1024,10)
r =  0.5001
     0.5631
     0.1281
     0.2028
     0.8263
     0.1940
     0.2823
     0.9742
     0.6837
     0.4811
```

为了使产生的随机数更均匀，β 尽量取得大一些。

- 取 $M = 2^L$，$a = 5^{2s+1}$，且满足 $5^{2s+1} < 2^L < 5^{2s+3}$。

常用的参数数值如表 13-1 所示。

表 13-1　乘同余法 2 产生随机数的常用参数取值表

L	s	a
21 ~ 25	4	1953125
26 ~ 30	5	48828125
31 ~ 34	6	1220703125
35 ~ 39	7	30517578125
40 ~ 44	8	762939453125
45 ~ 48	9	19073486328125

在 MATLAB 中编程实现的乘同余法 2 的函数为：MulMOD2

功能：用乘同余法 2 产生随机数列

调用格式：r = MulMOD2(L,s,x0,n)

其中，L：模的幂参数；

s：乘数的参数；
x0：随机数种子；
n：产生的随机数个数；
r：产生的随机数序列。

乘同余法 2 的 MATLAB 程序代码如下所示：

```
function r = MulMOD2(L,s,x0,n)
%模的幂参数：L
%乘数的参数：s
%随机数种子：x0
%产生的随机数个数：n
%产生的随机数序列：r
r = zeros(n,1);
x = zeros(n,1);
M = power(2,L);
a = power(5,2*s+1);
x(1) = x0;
r(1) = x(1)/M;
for i=2:n
    y = a*x(i-1);
    x(i) = mod(y,M);
    r(i) = x(i)/M;
end
```

例 13-4 乘同余法 2 产生随机数列应用实例。用乘同余法 2 产生 10 个随机数。

解： 在 MATLAB 中输入下列命令：

```
>> r=MulMOD2(28,5,34539223,10)
r =  0.1287
    0.0644
    0.1853
    0.8784
    0.2951
    0.4643
    0.1820
    0.2545
    0.5979
    0.2226
```

13.2.3 素数模同余法

素数模同余法不是取模 $M=2^L$，而是取模为小于 2^L 的最大素数。常用的两种素数模发生器介绍如下。

- 取 $L=35$，此时 $M=2^{35}-31$，适合的 $a=3125$，即

$$\begin{cases} x_n = 3125x_{n-1} \bmod (2^{35}-31) \\ r_n = x_n/(2^{35}-31) \end{cases}$$

- 取 $L=31$，此时 $M=2^{31}-1$，适合的 $a=16807$，即

$$\begin{cases} x_n = 16807 x_{n-1} \mod (2^{31}-1) \\ r_n = x_n / (2^{31}-1) \end{cases}$$

在 MATLAB 中编程实现的素数模同余法的函数为：PrimeMOD

功能：用素数模同余法产生随机数列

调用格式：r = PrimeMOD(x0,n,type)

其中，x0：随机数种子；

n：产生的随机数个数；

type：采用的公式类型；

r：产生的随机数序列。

素数模同余法的 MATLAB 程序代码如下所示：

```
function r = PrimeMOD(x0,n,type)
%随机数种子：x0
%产生的随机数个数：n
%采用的公式类型：type
%产生的随机数序列：r
function r = PrimeMOD(x0,n,type)
format long;
M1 = power(2,35)-31;
M2 = power(2,31)-1;
a1 = 3125;
a2 = 16807;
r = zeros(n,1);
x = zeros(n+5,1);
x(1) = x0;
if type == 1          %L 为 35 的素数同余法
    for i=2:n+5
        y = a1*x(i-1);
        x(i) = mod(y, M1);
    end
    r = x(6:(n+5))/M1;
else                  %L 为 31 的素数同余法
    for i=2:n+5
        y = a2*x(i-1);
        x(i) = mod(y, M2);
    end
    r = x(6:(n+5))/M2;
end
format short;
```

例 13-5 素数模同余法产生随机数列应用实例。用素数模同余法产生 10 个随机数。

解： 在 MATLAB 中输入下列命令：

```
>> r = PrimeMOD(2,10,1)
```

```
r =  0.7754
    0.1846
    0.8683
    0.5567
    0.5730
    0.6115
    0.0296
    0.4294
    0.9797
    0.6095
>> r = PrimeMOD(4,10,2)
r =  0.1311
    0.8758
    0.1882
    0.7155
    0.7172
    0.7388
    0.5340
    0.0777
    0.3239
    0.1383
```

13.3 产生指数分布的随机数列

指数分布的概率密度函数为

$$f(x)=\begin{cases}\dfrac{1}{\beta}e^{-\frac{x}{\beta}} & x\geqslant 0\\ 0 & \text{其他}\end{cases}$$

产生指数分布的随机数列的算法介绍如下：

❶ 产生[0,1]上均匀分布的随机数 r 。

❷ 计算 $x=-\beta\ln r$ ，则数列 $\{x\}$ 为均值为 β ，方差为 β^2 的指数分布随机数列。

在 MATLAB 中编程实现的指数分布的函数为：PowerDist

功能：产生指数分布的随机数列

调用格式：x = PowerDist(x0,n,beta)

其中，x0：随机数种子；

n：产生的随机数个数；

beta：指数分布参数；

x：产生的随机数序列。

产生指数分布的随机数列的 MATLAB 程序代码如下所示：

```
function x = PowerDist(x0,n,beta)
%随机数种子：x0
%产生的随机数个数：n
%指数分布参数：beta
```

```
%产生的随机数序列: x
format long;
x = zeros(n,1);
for i=1:n
    r = MixMOD(x0,2,1);       %用混合同余法产生随机数
    k = 0;
    while r(2) == 0           %防止产生坏的随机数
        k = k + 1;
        r(2) = power(2,k);
        r = MixMOD(r(2),2,1);
    end
    x(i) = -beta*log(r(2));
    x0 = x(i);
end
format short;
```

例 13-6 产生指数分布的随机数列应用实例。产生均值为 10 的指数分布的随机数列。

解： 在 MATLAB 中输入下列命令：

```
>> x =PowerDist(4,10,20)
x =  1.0913
     3.6977
     0.7623
     2.8843
    23.3382
    12.0544
     9.1600
     8.8944
    29.0235
     5.0186
    14.9359
    10.8064
    11.1164
     7.3646
    26.7678
     5.5437
    13.2125
     2.6454
    29.0697
     3.5730
```

可以检验上面产生的随机数的分布情况：

```
>> jz = mean(x)  %求数列 x 的均值
jz =  11.0480
>> fc = var(x)  %求数列 x 的方差
fc =  84.6023
```

检验的结果表明，产生的随机数序列的均值为 11.0480，方差为 84.6023，而两者的理

论值为 10 和 100，当产生的随机数个数越多的时候，随机数列越接近理论分布。

13.4 产生拉普拉斯分布的随机数列

拉普拉斯分布的概率密度函数为

$$f(x)=\frac{1}{2\beta}e^{-\frac{|x|}{\beta}}$$

产生拉普拉斯分布的随机数列的算法介绍如下：

❶ 产生[0,1]上均匀分布的随机数列 r_1 和 r_2。

❷ 计算

$$x=\begin{cases}-\beta\ln(1-r_2) & r_1\leqslant 0.5\\ \beta\ln r_2 & r_1>0.5\end{cases}$$

则数列 $\{x\}$ 为均值为 0，方差为 $2\beta^2$ 的拉普拉斯分布的随机数列。

在 MATLAB 中编程实现的拉普拉斯分布的函数为：LaplaceDist

功能：产生拉普拉斯分布的随机数列

调用格式：x = LaplaceDist(x0,x1,n,beta)

其中，x0：随机数种子；

x1：随机数种子

n：产生的随机数个数；

beta：拉普拉斯分布参数；

x：产生的随机数序列。

产生拉普拉斯分布的随机数列的 MATLAB 程序代码如下所示：

```
function x = LaplaceDist(x0,x1,n,beta)
%随机数种子：x0
%随机数种子：x1
%产生的随机数个数：n
%拉普拉斯分布参数：beta
%产生的随机数序列：x
format long;
x = zeros(n,1);
for i=1:n
    r = MixMOD(x0,10,1);                    %用混合同余法产生随机数
    u = MixMOD(x1,10,2);
    k = 0;
    while (r(10) == 0 || u(10) == 0)        %防止产生坏的随机数
        k = k + 1;
        r(10) = power(2,k);
        u(10) = power(2,power(2,k));
        r = MixMOD(r(10),2,1);
        u = MixMOD(u(10),2,2);
```

```
    end
    if r(10) <= 0.5
        x(i) = -beta*log(1-u(10));
    else
        x(i) = beta*log(u(10));
    end
    x0 = x1;
    x1 = x(i);
end
format short;
```

例 13-7 产生拉普拉斯分布的随机数列应用实例。产生均值为 0，方差为 8 的拉普拉斯分布的随机数列。

解：在 MATLAB 中输入下列命令：

```
>> x = LaplaceDist(23,671,2,20)
x =  -0.4220
      2.0087
     -1.0257
     -0.5790
     -0.3408
      0.6032
     -1.1900
      0.8356
     -0.9209
      0.4567
      0.1985
     -8.1181
     -2.5313
      1.8108
      1.7130
      4.2364
     -5.6092
      7.8711
      2.7976
      6.1938
```

可以检验上面产生的随机数的分布情况：

```
>> jz = mean(x) %求数列 x 的均值
jz =  0.3994
>> fc = var(x) %求数列 x 的方差
fc =  12.7462
```

从结果可以看出，方差有点偏大，说明数据分布有点分散。

13.5 产生瑞利分布的随机数列

瑞利分布的概率密度函数为

$$f(x)=\frac{x}{\sigma^2}e^{-\frac{x^2}{2\sigma^2}}\qquad x>0$$

产生瑞利分布的随机数列的算法介绍如下：

❶ 产生[0,1]上均匀分布的随机数 r 。

❷ 计算 $y=-2\ln r$ 。

❸ 计算 $x=\sigma\sqrt{y}$ ，则数列 $\{x\}$ 为均值为 $\sigma\sqrt{\frac{\pi}{2}}$ ，方差为 $(2-\frac{\pi}{2})\sigma^2$ 的瑞利分布随机数列。

在 MATLAB 中编程实现的瑞利分布的函数为：RelayDist

功能：产生瑞利分布的随机数列

调用格式：x = RelayDist(x0,sigma,n)

其中，x0：随机数种子；

sigma：瑞利分布的参数；

n：产生的随机数个数；

x：产生的随机数序列。

产生瑞利分布的随机数列的 MATLAB 程序代码如下所示：

```
function x = RelayDist(x0,sigma,n)
%随机数种子：x0
%瑞利分布的参数：sigma
%产生的随机数个数：n
%产生的随机数序列：x
format long;
x = zeros(n,1);
for i=1:n
    r = MixMOD(x0,10,1);        %用混合同余法产生随机数
    k = 0;
    while r(10) == 0             %防止产生坏的随机数
        k = k + 1;
        r(10) = power(2,k);
        r = MixMOD(r(10),2,1);
    end
    y = -2*log(r(10));
    x(i) = sigma*sqrt(y);
    x0 = x(i);
end
format short;
```

例 13-8 产生瑞利分布的随机数列应用实例。产生瑞利分布的随机数列，其中 $\sigma=2$ 。

解：在 MATLAB 中输入下列命令：

```
>> x = RelayDist(71,2,10)
x =  4.7972
```

```
    2.2460
    6.1131
    4.1889
    3.3208
    2.6432
    3.6000
    4.0933
    3.0495
    2.7259
```

13.6 产生柯西分布的随机数列

柯西分布的概率密度函数为

$$f(x)=\frac{\beta}{\pi[\beta^2+(x-\alpha)^2]} \quad \beta>0$$

产生柯西分布的随机数列的算法介绍如下：

❶ 产生 [0,1] 上均匀分布的随机数 r。

❷ 计算 $x=\alpha-\frac{\beta}{\tan(r\pi)}$，则数列 $\{x\}$ 为柯西分布的随机数列。

> 在 MATLAB 中编程实现的柯西分布的函数为：CauthyDist
> 功能：产生柯西分布的随机数列
> 调用格式：x = CauthyDist(x0,alpha,beta,n)
> 其中，x0：随机数种子；
> alpha：柯西分布的参数；
> beta：柯西分布的参数；
> n：产生的随机数个数；
> x：产生的随机数序列。

产生柯西分布的随机数列的 MATLAB 程序代码如下所示：

```
function x = CauthyDist(x0,alpha,beta,n)
%随机数种子：x0
%柯西分布的参数：alpha
%柯西分布的参数：beta
%产生的随机数个数：n
%产生的随机数序列：x
format long;
pi = 3.1415926535;
x = zeros(n,1);
for i=1:n
    r = MixMOD(x0,10,1);          %用混合同余法产生随机数
    k = 0;
    while r(10) == 0              %防止产生坏的随机数
       k = k + 1;
```

```
        r(10) = power(2,k);
        r = MixMOD(r(10),2,1);
    end
    y = tan(r(10)*pi);
    x(i) = alpha - beta/y;
    x0 = x(i);
end
format short;
```

例 13-9 产生柯西分布的随机数列应用实例。产生柯西分布的随机数列，其中 $\alpha=1$，$\beta=3$。

解：在 MATLAB 中输入下列命令：

```
>> x = CauthyDist(23,1,3,10)
x =  2.6907
    -0.1731
     5.1708
   -13.8144
     3.8492
     1.4792
    -2.6032
     2.3259
     4.2820
     2.9695
```

13.7 产生爱尔朗分布的随机数列

爱尔朗分布的概率密度函数为

$$f(x)=\begin{cases}\dfrac{\beta^{-m}x^{m-1}}{(m-1)!}e^{-\frac{x}{\beta}} & x\geqslant 0,\beta>0\\ 0 & x<0\end{cases}$$

产生爱尔朗分布的随机数列的算法介绍如下：

① 产生 m 个 $[0,1]$ 上均匀分布的相互独立的随机数 r_i。

② 通过变换 $x=-\beta\ln(\prod_{i=1}^{m}r_i)$，则数列 $\{x\}$ 为爱尔朗分布的随机数列。

在 MATLAB 中编程实现的爱尔朗分布的函数为：AELDist

功能：产生爱尔朗分布的随机数列

调用格式：x = AELDist(x0,m,beta,n)

其中，x0：随机数种子；

m：爱尔朗分布的参数；

beta：爱尔朗分布的参数；

n：产生的随机数个数；

x：产生的随机数序列。

产生爱尔朗分布的随机数列的 MATLAB 程序代码如下所示：

```
function x = AELDist(x0,m,beta,n)
%随机数种子：x0
%爱尔朗分布的参数：m
%爱尔朗分布的参数：beta
%产生的随机数个数：n
%产生的随机数序列：x
format long;
x = zeros(n,1);
u = zeros(m,1);
for i=1:n
    for j=1:m
        r = MixMOD(x0,10,1);           %用混合同余法产生随机数
        k = 0;
        while r(10) == 0               %防止产生坏的随机数
            k = k + 1;
            r(10) = power(2,k);
            r = MixMOD(r(10),2,1);
        end
        u(j) = r(10);
        x0 = u(j);
        x(i) = x(i) - beta*log(u(j));
    end
    x0 = x(i);
end
format short;
```

例 13-10 产生爱尔朗分布的随机数列应用实例。产生爱尔朗分布的随机数列，其中 $m=3$，$\beta=2$。

解： 在 MATLAB 中输入下列命令：

```
>> x = AELDist(21,3,2,10)
x =  1.3607
     3.9061
     8.4967
     5.0211
     7.6140
     6.6806
    14.8945
     5.1019
     6.3486
     4.1543
```

13.8 产生正态分布的随机数列

正态分布也叫高斯分布，正态分布的概率密度函数为

$$f(x)=\frac{1}{\sqrt{2\pi}\sigma}e^{-(x-\mu)^2/2\sigma^2}$$

产生正态分布的随机数列的算法介绍如下。

❶ 产生 12 个[0,1]上均匀分布的相互独立的随机数 r_i，令

$$y=\sum_{i=1}^{12}r_i-6$$

❷ 通过变换 $x=\mu+\sigma y$，则数列 $\{x\}$ 是均值为 μ，方差为 σ^2 的正态分布随机数列。

> 在 MATLAB 中编程实现的正态分布的函数为：GaussDist
>
> 功能：产生正态分布的随机数列
>
> 调用格式：x = GaussDist(x0,mu,sigma,n)
>
> 其中，x0：随机数种子；
>
> mu：正态分布的参数；
>
> sigma：正态分布的参数；
>
> n：产生的随机数个数；
>
> x：产生的随机数序列。

产生正态分布的随机数列的 MATLAB 程序代码如下所示：

```
function x = GaussDist(x0,mu,sigma,n)
%随机数种子：x0
%正态分布的参数：mu
%正态分布的参数：sigma
%产生的随机数个数：n
%产生的随机数序列：x
format long;
x = zeros(n,1);
u = zeros(12,1);
for i=1:n
    y = -6;
    for j=1:12
        r = MixMOD(x0,10,1);          %用混合同余法产生随机数
        k = 0;
        while r(10) == 0              %防止产生坏的随机数
            k = k + 1;
            r(10) = power(2,k);
            r = MixMOD(r(10),2,1);
        end
        u(j) = r(10);
        x0 = u(j);
        y = y + u(j);
    end
    x(i) = mu + sigma*y;
    x0 = x(i);
end
format short;
```

例 13-11 产生正态分布的随机数列应用实例。产生正态分布的随机数列，其中 $\mu=5$， $\sigma=2$。

解： 在 MATLAB 中输入下列命令：

```
>> x = GaussDist(31,5,2,10)
x =  3.4708
    4.2467
    7.4921
    6.1306
    3.3621
    2.4796
    6.4703
    4.7963
    2.2649
    4.6933
```

可以用如下的程序检验产生的随机数是否满足 $\mu=5$， $\sigma=2$ 的正态分布：

```
function numDist = test(x0,N)
x = GaussDist (31,5,2,10000);
mi = floor(min(x));
ma =  ceil(max(x));
numDist = zeros(ma-mi,1);
for i=1:N
   j = floor(x(i)) - mi;
   numDist(j+1) = numDist(j+1) + 1;
end
t=1:(ma-mi);
bar(t,numDist);
```

在 MATLAB 中输入下列命令：

```
>> num = test(41,10000)
num =    18
         40
        215
        380
        920
       1612
       1884
       1860
       1493
        882
        502
        154
         31
          4
```

而且可以从如图 13-3 所示的直方图中明显看出来：

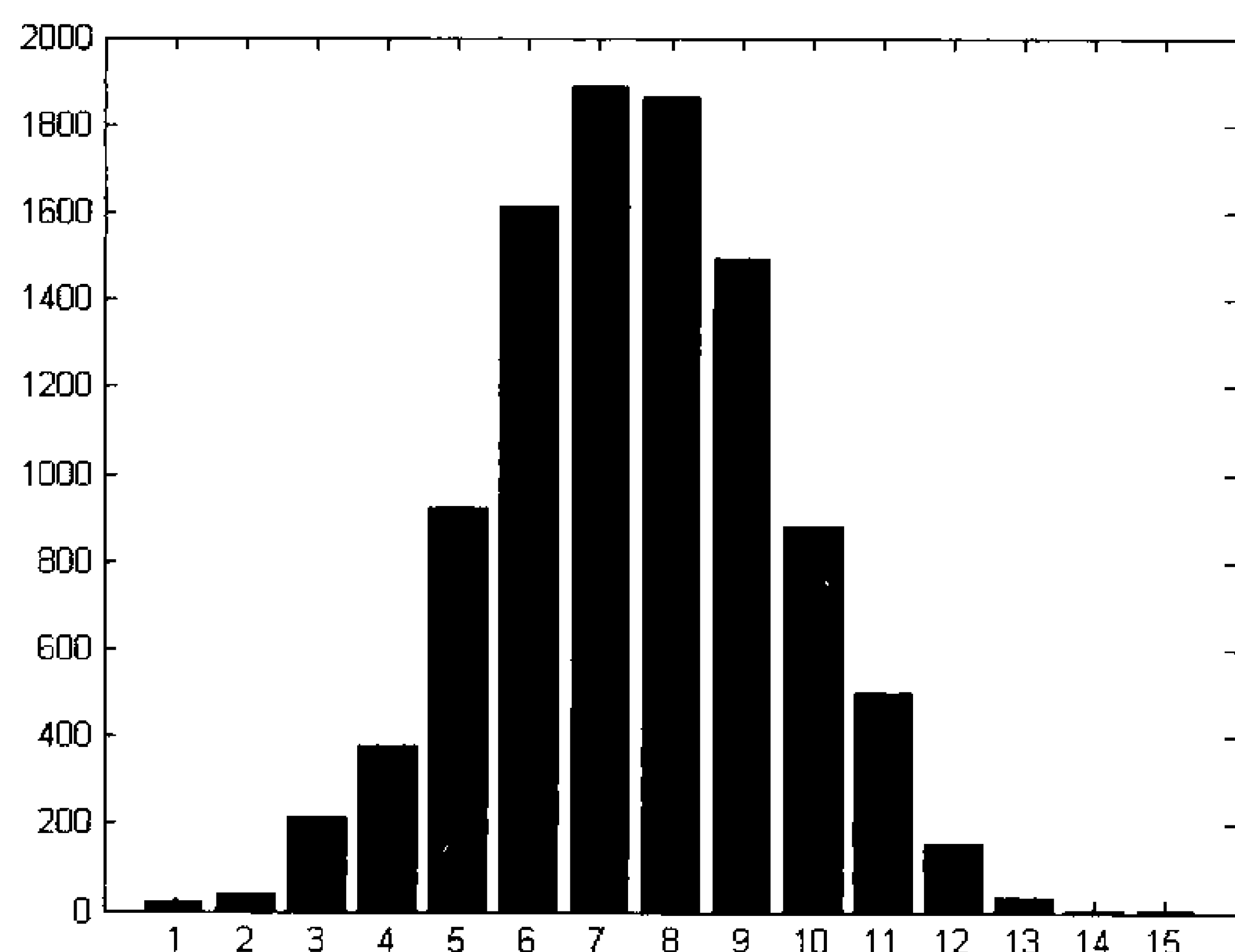


图 13-3　用正态分布法产生的随机数的分布图

13.9 产生韦伯分布的随机数列

韦伯分布的概率密度函数为

$$f(x)=\begin{cases}\dfrac{\alpha}{\beta^{\alpha}}x^{\alpha-1}e^{-(\frac{x}{\beta})^{\alpha}} & x\geqslant 0,\alpha,\beta>0\\ 0 & x<0\end{cases}$$

产生韦伯分布的随机数列的算法介绍如下：

❶ 产生[0,1]上均匀分布的随机数 r 。

❷ 计算 $x=\beta(-\ln r)^{1/\alpha}$ ，则数列 $\{x\}$ 为韦伯分布的随机数列。

> 在 MATLAB 中编程实现的韦伯分布的函数为：WBDist
>
> 功能：产生韦伯分布的随机数列
>
> 调用格式：x = WBDist (x0,alpha,beta,n)
>
> 其中，x0：随机数种子；
>
> alpha：韦伯分布的参数；
>
> beta：韦伯分布的参数；
>
> n：产生的随机数个数；
>
> x：产生的随机数序列。

产生韦伯分布的随机数列的 MATLAB 程序代码如下所示：

```
function x = WBDist(x0,alpha,beta,n)
%随机数种子：x0
%韦伯分布的参数：alpha
%韦伯分布的参数：beta
%产生的随机数个数：n
%产生的随机数序列：x
format long;
x = zeros(n,1);
```

```
for i=1:n
    r = MixMOD(x0,10,1);        %用混合同余法产生随机数
    k = 0;
    while r(10) == 0            %防止产生坏的随机数
        k = k + 1;
        r(10) = power(2,k);
        r = MixMOD(r(10),2,1);
    end
    t = 1/alpha;
    y = -log(r(10));
    x(i) = beta*power(y,t);
    x0 = x(i);
end
format short;
```

例 13-12 产生韦伯分布的随机数列应用实例。产生韦伯分布的随机数列，其中 $\alpha=2$，$\beta=2$。

解： 在 MATLAB 中输入下列命令：

```
>> x = WBDist(71,2,2,10)
x =  3.3921
     1.7214
     1.2474
     1.0062
     0.6637
     4.7105
     0.4223
     2.5681
     3.5539
     2.4819
```

13.10 产生泊松分布的随机数列

泊松分布的概率密度函数为

$$f(x)=\frac{\lambda^x e^{-\lambda}}{x!} \qquad x\in(0,1,\cdots,\lambda)$$

产生泊松分布的随机数列的算法介绍如下：

❶ 令 $b=1$，$i=0$。

❷ 产生 $[0,1]$ 上均匀分布的随机数 r。

❸ 令 $b=b\times r$。

❹ 如果 $b\geqslant e^{-\lambda}$，那么令 $i=i+1$，转❷。

❺ 取 $x=i$，则数列 $\{x\}$ 为泊松分布的随机数列。

在 MATLAB 中编程实现的泊松分布的函数为：PoisonDist
功能：产生泊松分布的随机数列

调用格式：x = PoisonDist (x0, lamda,n)

其中，x0：随机数种子；

lamda：泊松分布的参数；

n：产生的随机数个数；

x：产生的随机数序列。

产生泊松分布的随机数列的 MATLAB 程序代码如下所示：

```
function x =PoisonDist(x0,lamda,n)
%随机数种子：x0
%泊松分布的参数：lamda
%产生的随机数个数：n
%产生的随机数序列：x
format long;
x = zeros(n,1);
for i=1:n
    b = 1;
    tol = 1;
    k = 0;
    while tol == 1
        r = MixMOD(x0,10,1);          %用混合同余法产生随机数
        b = b*r(10);
        if b < exp(-lamda)
            tol = 0 ;
            x(i) = k;
        else
            k = k+1;
        end
    end
    x0 = x0 + 31;
end
format short;
```

例 13-13 产生泊松分布的随机数列应用实例。产生泊松分布的随机数列，其中 $\lambda = 2$。

解： 在 MATLAB 中输入下列命令：

```
>> x = PoisonDist(11,2,10)
x =  2
     0
     5
    13
    76
    26
     1
   197
     1
    55
```

13.11 产生贝努里分布的随机数列

贝努里分布的概率密度函数为：

$$f(x)=\begin{cases}p & x=1\\ 1-p & x=0\end{cases}$$

其中 $0\leqslant p\leqslant 1$。

产生贝努里分布的随机数列的算法介绍如下：

❶ 产生 $[0,1]$ 上均匀分布的随机数 r。

❷ 如果 $r\leqslant p$，则令 $x=1$，否则令 $x=0$。

> 在 MATLAB 中编程实现的贝努里分布的函数为：BenuliDist
>
> 功能：产生贝努里分布的随机数列
>
> 调用格式：x = BenuliDist (x0, p,n)
>
> 其中，x0：随机数种子；
>
> p：贝努里分布的参数；
>
> n：产生的随机数个数；
>
> x：产生的随机数序列。

产生贝努里分布的随机数列的 MATLAB 程序代码如下所示：

```
function x =BenuliDist(x0,p,n)
%随机数种子：x0
%贝努里分布的参数：p
%产生的随机数个数：n
%产生的随机数序列：x
format long;
x = zeros(n,1);
for i=1:n
    r = MixMOD(x0,10,1);        %用混合同余法产生随机数
    if r(10) <= p
        x(i) = 1;
    else
        x(i) = 0;
    end
    x0 = 2*x0;
end
format short;
```

例 13-14 产生贝努里分布的随机数列应用实例。产生贝努里分布的随机数列，其中 $p=0.5$。

解： 在 MATLAB 中输入下列命令：

```
>> x = BenuliDist(33,0.5,10)
```

```
x =  1
     1
     1
     1
     0
     1
     0
     0
     0
     1
```

13.12 产生贝努里-高斯分布的随机数列

贝努里-高斯分布的随机变量为贝努里分布与正态分布的随机变量的乘积。

产生贝努里-高斯分布的随机数列的算法介绍如下：

1. 产生贝努里分布的随机数 y。
2. 产生正态分布的随机数 z。
3. 计算 $x = y \times z$，则数列 $\{x\}$ 为贝努里-高斯分布的随机数列。

在 MATLAB 中编程实现的贝努里-高斯分布的函数为：BGDist

功能：产生贝努里-高斯分布的随机数列

调用格式：x = BGDist(x0,p,mu,sigma,n)

其中，x0：随机数种子；

p：贝努里-高斯分布的参数；

mu：贝努里-高斯分布的参数；

sigma：贝努里-高斯分布的参数；

n：产生的随机数个数；

x：产生的随机数序列。

产生贝努里-高斯分布的随机数列的 MATLAB 程序代码如下所示：

```
function x =BGDist(x0,p,mu,sigma,n)
%随机数种子：x0
%贝努里-高斯分布的参数：p
%贝努里-高斯分布的参数：mu
%贝努里-高斯分布的参数：sigma
%产生的随机数个数：n
%产生的随机数序列：x
format long;
x = zeros(n,1);
for i=1:n
    y = BenuliDist(x0,p,10);
    z = GaussDist(2*x0,mu,sigma,10);
    x(i) = y(10)*z(10);
    x0 = 2*x0;
```

```
end
format short;
```

例 13-15 产生贝努里-高斯分布的随机数列应用实例。产生贝努里-高斯分布的随机数列，其中 $p=0.3$ ，$\mu=1$ ，$\sigma=2$ 。

解： 在 MATLAB 中输入下列命令：

```
>> x = BGDist(85,0.3,1,2,12)
x =      0
         0
         0
    0.5661
         0
    1.8039
   -0.0869
         0
   -1.3963
         0
         0
         0
```

13.13 产生二项式分布的随机数列

二项式分布的概率密度函数为：

$$f(x)=C_n^x p^x(1-p)^{n-x} \quad x\in(0,1,2,\cdots,n)$$

其中 $C_n^x=\dfrac{n!}{x!(n-x)!}$，$0\leqslant p\leqslant 1$ 。

产生二项式分布的随机数列的算法介绍如下：

❶ 产生贝努里分布的随机数 $r_1,r_2,\cdots,r_n$ 。

❷ 令 $x=\sum_{i=1}^{n} y_i$ ，则数列 $\{x\}$ 为二项式分布的随机数列。

> 在 MATLAB 中编程实现的二项式分布的函数为：TwoDist
> 功能：产生二项式分布的随机数列
> 调用格式：x = TwoDist (x0,p,n,N)
> 其中，x0：随机数种子；
> p：二项式分布的参数；
> n：二项式斯分布的参数；
> N：产生的随机数个数；
> x：产生的随机数序列。

产生二项式分布的随机数列的 MATLAB 程序代码如下所示：

```
function x =TwoDist(x0,p,n,N)
```

```
%随机数种子：x0
%二项式分布的参数：p
%二项式分布的参数：n
%产生的随机数个数：N
%产生的随机数序列：x
format long;
x = zeros(N,1);
for i=1:N
    y = BenuliDist(x0,p,n);
    x(i) = sum(y);
    x0 = 2*x0;
end
format short;
```

例 13-16 产生二项分布的随机数列应用实例。产生二项式分布的随机数列，其中 $p=0.5$， $n=10$。

解： 在 MATLAB 中输入下列命令：

```
>> x =TwoDist(31,0.5,10,10)
x =  7
     8
     7
     8
     8
     7
     7
     6
     6
     6
```

13.14 小结

几乎每一种编程语言都提供自己的随机数生成函数，不管它们的性能如何，随机数的基本算法还是同余法，因为同余法不但实现简单，而且产生的随机数序列均匀性好。

第 14 章　特殊函数计算

特殊函数是指一些类似三角函数及伽玛函数、贝塞尔函数等超几何数列函数，具有特殊的性质和特点，在现实中得到大量的运用的函数。对这些函数的研究并不在一般数学分析或实函数分析范畴之内，传统上对特殊函数的分析主要基于对其的数值展开基础上。

通过本章的学习，读者不仅能掌握数理方程中常用的特殊函数的算法，而且还能熟练使用 MATLAB 编程来实现这些算法。

14.1 伽玛函数和贝塔函数

伽玛函数的定义如下：

$$\Gamma(x)=\int_0^{\infty}\mathrm{e}^{-t}t^{x-1}\mathrm{d}t \qquad (x>0)$$

伽玛函数 $\Gamma(x)$ 的一般算法介绍如下。

❶ 如果 $2<x\leqslant 3$，转❷；否则

a）如果 $0<x\leqslant 1$，用公式 $\Gamma(x)=\dfrac{\Gamma(x+2)}{x(x+1)}$ 进行变换；

b）如果 $1<x\leqslant 2$，用公式 $\Gamma(x)=\dfrac{\Gamma(x+1)}{x}$ 进行变换；

c）如果 $x>3$，用公式 $\Gamma(x)=(x-1)(x-2)\cdots(x-i)\Gamma(x-i)$ 进行变换，直到 $2<x-i\leqslant 3$。即 i 是满足下面不等式的整数：

$$x-3\leqslant i\leqslant -2$$

❷ 计算 $\Gamma(x)=\sum\limits_{i=0}^{10}c_i(x-2)^{10-i}$，其中

$$c_0=0.0000677106, c_1=-0.0003442342$$
$$c_2=0.0015397681, c_3=-0.0024467480$$
$$c_4=0.0109736958, c_5=-0.0002109075$$
$$c_6=0.0742379071, c_7=0.0815782188$$
$$c_8=0.4118402518, c_9=0.4227843370$$
$$c_{10}=1.0000000000$$

算法结束。

> 在 MATLAB 中编程实现的计算伽玛函数值的逼近法函数为：gamafun
> 功能：用逼近法计算伽玛函数的值

调用格式：function gx = gamafun(x)
其中，x：自变量的值；
gx：自变量取 x 值时的伽玛函数值。

计算伽玛函数值的逼近法的 MATLAB 程序代码如下所示：

```
function gx = gamafun(x)
%自变量的值：x
%自变量取 x 值时的伽玛函数值：gx
format long;
if x < 0
    disp('x 必须大于 0！');
    return;
else
    if x<= 1
        gx = gf(x+2)/x/(x+1);      %公式变换
    else
        if x<=2
            gx = gf(x+1)/x;        %公式变换
        else
            if x<=3
                gx = gf(x);
            else                   %公式变换
                k = floor(x) - 2;
                c = 1;
                for i=1:k
                    c = c*(x-i);
                end
                gx = c*gf(x-k);
            end
        end
    end
end
function tx = gf(x)          %伽玛函数的多项式逼近
format long;
c = [0.0000677106;-0.0003442342;0.0015397681;
    -0.0024467480;0.0109736958;-0.0002109075;
    0.0742379071;0.0815782188;0.4118402518;
    0.4227843370;1.00000000];
tx = 0;
for i=1:11
    tx = tx + c(i)*power(x-2,11-i);
end
```

伽玛函数 $\Gamma(x)$ 的 Lanczos 算法介绍如下。

$$\Gamma(x+1) \approx (x+\gamma+0.5)^{x+0.5} e^{-(x+\gamma+0.5)} \times \sqrt{2\pi}\left(a_0 + \frac{a_1}{x+1} + \frac{a_2}{x+2} + \cdots + \frac{a_N}{x+N}\right)$$

一般情况下，采用的是 $\gamma = 5$，$N = 6$，系数为：

$$a_0 = 1.000000000000000, a_1 = 76.18009172947146$$
$$a_2 = -86.50532032941677, a_3 = 24.01409824083091$$
$$a_4 = -1.231739572450155, a_5 = 0.1208650973866179e-2$$
$$a_6 = -0.5395239384953e-5$$

在 MATLAB 中编程实现的计算伽玛函数的自然对数的 Lanczos 算法函数为：lngama
功能：用 Lanczos 算法计算伽玛函数的自然对数值
调用格式：function Tx = lngama(x)
其中，x：自变量的值；
　　Tx：自变量取 x 值时的伽玛函数值的自然对数。

计算伽玛函数值的 Lanczos 算法的 MATLAB 程序代码如下所示：

```
function Tx = lngama(x)
%自变量的值：x
%自变量取 x 值时的伽玛函数值的自然对数：Tx
if x < 0
   disp('x 必须大于 0！');
   return;
end
format long;
c = [76.18009173;-86.50532033;24.01409822;
   -1.231739516;0.00120858003;-0.00000536382];    %Lanczos 算法中的系数
const1 = sqrt(2*pi);
T1 = (x-0.5)*log(x+4.5);
T2 = 1;
for i=1:6
   T2 = T2 + c(i)/(x-1+i);
end
Tx = T1 - x - 4.5 + log(const1*T2);
Tx = vpa(Tx,8);
```

由伽玛函数计算阶乘的公式如下所示：

$$n! = \Gamma(n+1)$$

在 MATLAB 中编程实现的由伽玛函数计算阶乘的函数为：factbygama
功能：用伽玛函数求阶乘
调用格式：function y= factbygama (n)
其中，n：自然数；
　　y：n 的阶乘。

伽玛函数计算阶乘的 Lanczos 算法的 MATLAB 程序代码如下：

```
function y = factbygama(n)
%自然数：n
%阶乘值：y
format long;
```

```
y = 1;
if n <= 32
    for i=1:n
        y = y*i;
    end
else
    y = exp(lngama(n+1));    %先求阶乘的自然对数，再求阶乘
end
```

贝塔函数的定义如下：

$$B(x,y)=\int_0^1 t^{x-1}(1-t)^{y-1}\mathrm{d}t(x,y>0)$$

贝塔函数的计算方法为：

$$B(x,y)=\frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$$

在 MATLAB 中编程实现的计算贝塔函数值的函数为：Beta

功能：用伽玛函数计算贝塔函数的值

调用格式：BF = Beta(x,y)

其中，x：自变量的值；

　　y：第二个自变量的值；

　　BF：贝塔函数值。

计算贝塔函数值的 MATLAB 程序代码如下所示：

```
function BF = Beta(x,y)
%自变量的值：x
%第二个自变量的值：y
%贝塔函数值：BF
if x <= 0 || y <= 0
    disp('x,y 必须大于 0! ');
    return;
end
format long;
Bx = exp(lngama(x));
By = exp(lngama(y));
Bxy = exp(lngama(x+y));
BF = Bx*By/Bxy;
```

例 14-1　伽玛函数应用实例 1。计算 $x=0.1,0.2,\cdots,1$ 时的伽玛函数值，并与 MATLAB 自带的伽玛函数 Gamma 相比较。

解：在 MATLAB 命令行输入下列命令

```
>> for i=1:10
      my(i) = gamafun(i/10);
      sy(i) = Gamma(i/10);
      ly(i) = exp(lngama(i/10));
end
```

将结果列成表格比较如下：

x	gamafun(x)	exp(lngama(x))	Gamma(x)
0.1	9.513507698750150	9.513507698226600	9.513507698668730
0.2	4.590843713141700	4.590843711763810	4.590843711998800
0.3	2.991568998925080	2.991568987520560	2.991568987687590
0.4	2.218159602303030	2.218159543623710	2.218159543757690
0.5	1.772454059230470	1.772453850790530	1.772453850905520
0.6	1.489192831970880	1.489192248709680	1.489192248812820
0.7	1.298056716682360	1.298055332552100	1.298055332647560
0.8	1.164232626305490	1.164229713634820	1.164229713725300
0.9	1.068634296052830	1.068628702031910	1.068628702119320
1.0	1.000009999750000	0.999999999914230	1.000000000000000

MATLAB 系统采用的是更精确的算法，但是本节中的两个逼近函数都达到了10^{-6}的精度，在一般的计算中已经足够。

例 14-2 伽玛函数应用实例 2。计算$n=30,31,\cdots,38$时的阶乘，并与 MATLAB 自带的阶乘函数 factorial 相比较。

解：在 MATLAB 命令行输入下列命令：

```
>> for i=1:9
        mf(i) = factbygama(i+29);
        sf(i) = factorial(i+29);
end
```

将结果列成表格比较如下：

n	factbygama(n)	factorial(n)
30	2.652528598121910e+032	2.652528598121910e+032
31	8.222838654177922e+033	8.222838654177922e+033
32	2.631308369336935e+035	2.631308369336935e+035
33	8.683317617196106e+036	8.683317618811886e+036
34	2.952327989852689e+038	2.952327990396041e+038
35	1.033314796450539e+040	1.033314796638614e+040
36	3.719933267229548e+041	3.719933267899012e+041
37	1.376375308877653e+043	1.376375309122634e+043
38	5.230226173745393e+044	5.230226174666010e+044

从上表的比较结果可以看出，当$n \leqslant 32$时，函数 factbygama 计算得到的阶乘结果与 MATLAB 系统得到的结果是一样的，当$n>32$时，两者算出的结果有差别，可以预计 MATLAB 系统在$n \leqslant 32$时可能采用本书中的方法计算阶乘，而当$n>32$时采用了其他的方法。

例 14-3 贝塔函数应用实例。计算

$$\begin{cases} x = 1.2, 1.4, 1.6 \\ y = 2.2, 2.4, 2.6 \end{cases}$$

时的贝塔函数值，并与 MATLAB 自带的贝塔函数 beta 相比较。

解：在 MATLAB 命令行输入下列命令：

```
>> for i=1:3
       for j=1:3
           mb(i,j) = Beta(i*0.2+1,j*0.2+2);
           sb(i,j) = beta(i*0.2+1,j*0.2+2);
       end
   end
```

将结果列成表格比较如下：

x \ y	2.2		2.4		2.6	
	Beta	beta	Beta	beta	Beta	beta
1.2	0.339339335353010	0.339339335353010	0.306837165965250	0.306837165965250	0.279630990623950	0.279630990623950
1.4	0.263003285113070	0.263003285113070	0.234787177953980	0.234787177953980	0.211409023948420	0.211409023948420
1.6	0.209723242967960	0.209723242967960	0.184982895954870	0.184982895954870	0.164682559636530	0.164682559636530

可以看出，两者结果完全一样。

14.2 不完全伽玛函数

不完全伽玛函数的定义如下所示：

$$\Gamma(\alpha,x)=\frac{P(\alpha,x)}{\Gamma(\alpha)},\alpha>0,x>0$$

其中

$$P(\alpha,x)=\int_0^x \mathrm{e}^{-t}t^{\alpha-1}\mathrm{d}t$$

不完全伽玛函数 $\Gamma(\alpha,x)$ 的算法介绍如下。

❶ 如果 $x<\alpha+1$，有

$$P(\alpha,x)=\mathrm{e}^{-x}x^{\alpha}\sum_{k=0}^{\infty}\frac{\Gamma(\alpha)x^k}{\Gamma(\alpha+1+k)}$$

❷ 如果 $x\geqslant\alpha+1$，有

$$\Gamma(\alpha,x)=1-\frac{Q(\alpha,x)}{\Gamma(\alpha)} \qquad \alpha>0,x>0$$

$$Q(\alpha,x)=\cfrac{e^{-x}x^{\alpha}}{x+\cfrac{1-\alpha}{1+\cfrac{1}{x+\cfrac{2-\alpha}{1+\cfrac{2}{x+\cdots+\cfrac{n-\alpha}{1+\cfrac{n}{x+\cdots}}}}}}}$$

在 MATLAB 中编程实现的计算不完全伽玛函数值的函数为：gamap

功能：用逼近法计算不完全伽玛函数的值

调用格式：function gp = gamap(x,alpha)

其中，x：自变量的值；

alpha：参数；

gp：自变量取 x 值时的不完全伽玛函数值。

计算不完全伽玛函数值的逼近法的 MATLAB 程序代码如下所示：

```
function gp = gamap(x,alpha)
%自变量的值：x
%参数：alpha
%自变量取 x 值时的不完全伽玛函数值：gp
function gp = gamap(x,alpha)
format long;
if  x <=0  || alpha<=0
    disp('x 和 alpha 不能小于 0!');
    return;
end
c = exp(-x)*power(x,alpha);
if x<alpha+1
    gp = c/exp(lngama(alpha+1));
    tol = 1;
    i = 1;
    while tol == 1
        T1 = exp(lngama(alpha+1+i));
        dP = c*power(x,i)/T1;
        if dP/gp < 1.0e-20        %精度控制
            tol = 0;
        else
            gp = gp + dP;
        end
        i = i + 1;
    end
else
    tmp1 = gamafun(alpha);
    n = 10;
    while 1
        t1 = lfs(n,x,alpha);
```

```
        t2 = lfs(n+1,x,alpha);
        if abs(t2-t1)< 1.0e-20
            gp = 1 - c * t1/tmp1;
            break;
        else
            n = n + 1;
        end
    end
end
function q = lfs(n,x,alpha)    %连分数求值函数
format long;
q = n;
for i=n:-1:2
    q1 = x + (i-alpha)/(1 + q);
    q = 1 + (i-1)/q1;
end
q = x + (1-alpha)/q;
q = 1/q;
```

例 14-4 不完全伽玛函数应用实例。计算 $x=1.1,1.2,\cdots,2$，且 $\alpha=0.5$ 时的不完全伽玛函数值，并与 MATLAB 自带的不完全伽玛函数 gammainc 相比较。

解：在 MATLAB 命令行输入下列命令：

```
>> for i=1:10
     mp(i) = gamap(1+i/10,0.5);
     sp(i) = gammainc(1+i/10,0.5);
end
```

将结果列成表格比较如下，经过比较可以看出本方法还是有一定的误差。

x	gamap(x)	gammainc (x)
1.1	0.861989262512380	0.861989262431340
1.2	0.878664749726410	0.878664749641520
1.3	0.893136285095220	0.893136285006620
1.4	0.905735693250950	0.905735693158790
1.5	0.915955963174010	0.916735483336450
1.6	0.925738289144700	0.926361729879690
1.7	0.934302367600680	0.934803580921870
1.8	0.941815570182830	0.942220428876400
1.9	0.948418973856030	0.948747417142630
2	0.954232226443370	0.954499736103640

14.3 不完全贝塔函数

不完全贝塔函数的定义如下所示：

$$B_x(\alpha,\beta)=\frac{1}{B(\alpha,\beta)}\int_0^x t^{\alpha-1}(1-t)^{\beta-1}\mathrm{d}t$$

$$(\alpha,\beta>0,0\leqslant x\leqslant 1)$$

不完全贝塔函数 $B_x(\alpha,\beta)$ 的算法介绍如下。

❶ 当 $x<\dfrac{\alpha+1}{\alpha+\beta+2}$ 时，

$$B_x(\alpha,\beta)=\frac{x^{\alpha}(1-x)^{\beta}}{\alpha B(\alpha,\beta)}\phi(x)$$

$$\phi(x)=\cfrac{1}{1+\cfrac{s_1}{1+\cfrac{s_2}{1+\cdots}}}$$

其中

$$s_{2n-1}=-\frac{(\alpha+n)(\alpha+\beta+n)x}{(\alpha+2n)(\alpha+2n+1)}$$

$$s_{2n}=\frac{n(\beta-n)x}{(\alpha+2n-1)(\alpha+2n)}$$

❷ 当 $x\geqslant\dfrac{\alpha+1}{\alpha+\beta+2}$ 时，用公式 $B_x(\alpha,\beta)=1-B_{1-x}(\beta,\alpha)$ 进行变换，再转❶进行计算。

在 MATLAB 中编程实现的计算不完全贝塔函数值的函数为：betap

功能：用逼近法计算不完全贝塔函数的值

调用格式：function bp = betap(x,a,b)

其中，x：自变量的值；

a：第一个参数；

b：第二个参数；

bp：自变量取 x 值时的不完全贝塔函数值。

计算不完全贝塔函数值的逼近法的 MATLAB 程序代码如下所示：

```
function bp = betap(x,a,b)
%自变量的值：x
%第一个参数：a
%第二个参数：b
%自变量取 x 值时的不完全贝塔函数值：bp
format long;
if  a <=0 || b<=0
   disp('a 和 b 不能小于 0!');
   return;
end
if  x == 0 || x == 1
   bp = 0;
end
c1 = power(x,a);
```

```
c2 = power(1-x,b);
c3 = Beta(a,b);
if x<((a+1)/(a+b+2))
    n = 1;
    while 1
        f1 = fi(2*n,x,a,b);
        f2 = fi(2*n+2,x,a,b);
        if abs(f2-f1)<1.0e-30        %连分数值的精度控制
            bp  = f2*c1*c2/a/c3;
            break;
        else
            n = n + 1;
        end
    end
else
    if x ==0.5 && a == b
        bp = 0.5;
    else
        n = 1;
        while 1
            f1 = fi(2*n,1-x,b,a);
            f2 = fi(2*n+2,1-x,b,a);
            if abs(f2-f1)<1.0e-30     %连分数值的精度控制
                bp  = 1-f2*c1*c2/b/c3;
                break;
            else
                n = n + 1;
            end
        end
    end
end
function F = fi(N,x,a,b)           %连分数求值函数
format long;
n = N/2;
F = 0;
for i=n:-1:1
    tmpU = (a+2*i-1)*(a+2*i);
    s2 = i*(b - i)*x/tmpU;
    F1 = s2/(1+F);
    tmpV = (a+2*i-2)*(a+2*i-1);
    s1 = -(a+i-1)*(b+a+i-1)*x/tmpV;
    F = s1/(1+F1);
end
F = 1/(1+F);
```

例 14-5 不完全贝塔函数应用实例。验证不完全贝塔函数值，并与 MATLAB 自带的不完全贝塔函数 betainc 相比较，其中参数的取值如下表所示：

a	b	x
0.5	0.5	0.1
0.5	1	0.5
1	0.2	0.1
1	1	0.01
1	2	0.4
5	0.5	0.9
5	1	0.1
10	2	0.3
10	5	0.5
20	10	0.7
20	10	0.2
50	8	0.4
50	20	0.6
100	50	0.8

解：在 MATLAB 命令行输入下列命令：

```
>> a = [0.5;0.5;1;1;1;5;5;10;10;20;20;50;50;100];
>> b = [0.5;1;0.2;1;2;0.5;1;2;5;10;10;8;20;50];
>> x = [0.1;0.5;0.1;0.01;0.4;0.9;0.1;0.3;0.5;0.7;0.2;0.4;0.6;0.8];
>> for i=1:14
mbp(i) = betap(x(i),a(i),b(i));
sbp(i) = betainc(x(i),a(i),b(i));
end
```

将结果列成表格比较如下：

betap	betainc
0.204832764699130	0.204832764699130
0.707106781186550	0.707106781186550
0.020851637639020	0.020851637639020
0.010000000000000	0.010000000000000
0.640000000000000	0.640000000000000
0.316642915020010	0.316642915020010
0.000010000000000	0.000010000000000
0.000047239200000	0.000047239200000
0.089782714843750	0.089782714843750
0.635995918928060	0.635995918928060
0.000000015753450	0.000000015753450
0.000000000000100	0.000000000000100
0.021479401354680	0.021479401354680
0.999926936386330	0.999926936386330

从结果可以看出，betap 得出的结果和 MATLAB 得出的结果是完全一样的。

14.4 第一类整数阶贝塞尔函数

第一类 n 阶（ n 为整数）贝塞尔函数的积分表达式为：

$$J_n(x)=\frac{1}{2\pi}\int_{-\pi}^{\pi}\cos(nt-x\sin t)\mathrm{d}t$$

递推关系为：

$$J_{n+1}(x)=\frac{2n}{x}J_n(x)-J_{n-1}(x)$$

其中 $J_0(x)$ 与 $J_1(x)$ 的计算公式如下：

❶ 当 $|x|<8$ 时，有：

$$J_0(x)=\frac{A_0+A_1x^2+A_2x^4+A_3x^6+A_4x^8+A_5x^{10}}{B_0+B_1x^2+B_2x^4+B_3x^6+B_4x^8+B_5x^{10}}$$

$$J_1(x)=\frac{x(C_0+C_1x^2+C_2x^4+C_3x^6+C_4x^8+C_5x^{10})}{D_0+D_1x^2+D_2x^4+D_3x^6+D_4x^8+D_5x^{10}}$$

其系数的具体值如下：

$$A_0=57568490574.0,\quad A_1=-13362590354.0,$$
$$A_2=651619640.7,\quad A_3=-11214424.18,$$
$$A_4=77392.33017,\quad A_5=-184.9052456$$
$$B_0=57568490411.0,\quad B_1=1029532985.0,$$
$$B_2=9494680.718,\quad B_3=59272.64853,$$
$$B_4=267.8532712,\quad B_5=1.0$$
$$C_0=72362614232.0,\quad C_1=-7895059235.0,$$
$$C_2=242396853.1,\quad C_3=-2972611.439,$$
$$C_4=15704.4826,\quad C_5=-30.16036606$$
$$D_0=144725228443.0,\quad D_1=2300535178.0,$$
$$D_2=18583304.74,\quad D_3=99447.43394,$$
$$D_4=376.9991397,\quad D_5=1.0$$

❷ 当 $|x|\geqslant 8$ 时，有：

$$J_0(x)=\sqrt{\frac{2}{\pi|x|}}[R(z)\cos\theta-zS(z)\sin\theta],$$

$$\begin{cases}J_1(x)=\sqrt{\dfrac{2}{\pi|x|}}[P(z)\cos\varphi-zQ(z)\sin\varphi],x>0\\ J_1(-x)=-J_1(x),x<0\end{cases},$$

其中 $z=\dfrac{8}{|x|},\theta=|x|-\dfrac{\pi}{4},\varphi=|x|-\dfrac{3\pi}{4}$，且有：

$$R(z) = r_0 + r_1 z^2 + r_2 z^4 + r_3 z^6 + r_4 z^8$$
$$S(z) = s_0 + s_1 z^2 + s_2 z^4 + s_3 z^6 + s_4 z^8$$
$$P(z) = p_0 + p_1 z^2 + p_2 z^4 + p_3 z^6 + p_4 z^8$$
$$Q(z) = q_0 + q_1 z^2 + q_2 z^4 + q_3 z^6 + q_4 z^8$$

其系数的具体值如下：

$r_0 = 1.0$,　　$r_1 = -0.001098628627$,

$r_2 = 0.00002734510407$,　　$r_4 = -0.000002073370639$,

$r_5 = 0.0000002093887211$

$s_0 = -0.01562499995$,　　$s_1 = 0.0001430488765$,

$s_2 = -0.6911147651\times10^{-5}$,　　$s_3 = 0.7621095161\times10^{-6}$,

$s_4 = -0.934935152\times10^{-7}$

$p_0 = 1.0$,　　$p_1 = 0.00183105$,

$p_2 = -0.3516396496\times10^{-4}$,　　$p_3 = 0.2457520174\times10^{-5}$,

$p_4 = -0.240337019\times10^{-6}$

$q_0 = 0.04687499995$,　　$q_1 = -0.0002002690873$,

$q_2 = 0.8449199096\times10^{-5}$,　　$q_3 = -0.88228987\times10^{-6}$,

$q_4 = 0.105787412\times10^{-6}$

在 MATLAB 中编程实现的计算第一类整数阶贝塞尔函数值的函数为：bessel

功能：用逼近法计算伽玛函数的值

调用格式：function Jx = bessel(n,x)

其中，n：第一类贝塞尔函数的阶；

　　x：自变量的值；

　　Jx：自变量取 x 值时的第一类整数阶贝塞尔函数的值。

求第一类整数阶贝塞尔函数值的 MATLAB 程序代码如下所示：

```
function Jx = bessel(n,x)
%第一类贝塞尔函数的阶：n
%自变量的值：x
%自变量取 x 值时的第一类整数阶贝塞尔函数的值：Jx
format long;
if n == 0     %零阶贝塞尔函数
    a = [5.7568490574e10;-1.3362590354e10;6.516196407e8;
        -1.211442418e7;7.739233017e4;-1.849052456e2];
    b = [5.7568490411e10;1.029532985e9;9.494680718e6;
        5.927264853e4;2.678532712e2;1.0];
    r =[1.0;-0.1098628627e-2;0.2734510407e-4;
       -0.2073370639e-5;0.2093887211e-6];
    s =[-0.1562499995e-1;0.1430488765e-3;-0.6911147651e-5;
       0.7621095161e-6;-0.934935152e-7];
    if abs(x) < 8       %自变量绝对值小于 8 的计算公式
```

```
            J0_u = a(1);
            J0_d = b(1);
            for i=2:6
                J0_u = J0_u + a(i)*power(x,2*(i-1));
                J0_d = J0_d + b(i)*power(x,2*(i-1));
            end
            J0 = J0_u/J0_d;
        else                    %自变量绝对值不小于 8 的计算公式
            z = 8/abs(x);
            sita = abs(x) - pi/4;
            R0 = r(1);
            S0 = s(1);
            for i=2:5
                R0 = R0 + r(i)*power(z,2*(i-1));
                S0 = S0 + s(i)*power(z,2*(i-1));
            end
            J0 = (R0*cos(sita)-z*S0*sin(sita))*sqrt(2/pi/abs(x));
        end
        Jx = J0;
    else
        if n == 1     %一阶贝塞尔函数
            c = [7.2362614232e10;-7.895059235e9;2.423968531e8;
                -2.972611439e6;1.570448260e4;-3.016036606e1];
            d = [1.44725228443e11;2.300535178e9;1.858330474e7;
                9.944743994e4;3.769991397e2;1.0];
            p = [1.0;0.183105e-2;-0.3516396496e-4;
                0.2457520174e-5;-0.240337019e-6];
            q = [0.4687499995e-1;-0.2002690873e-3;
                0.8449199096e-5;-0.88228987e-6;
                0.105787412e-6];
            if abs(x) < 8      %自变量绝对值小于 8 的计算公式
                J1_u = c(1);
                J1_d = d(1);
                for i=2:6
                    J1_u = J1_u + c(i)*power(x,2*(i-1));
                    J1_d = J1_d + d(i)*power(x,2*(i-1));
                end
                J1 = x*J1_u/J1_d;
            else                    %自变量绝对值不小于 8 的计算公式
                z = 8/abs(x);
                fi = abs(x) - 3*pi/4;
                P1 = p(1);
                Q1 = q(1);
                for i=2:5
                    P1 = P1 + p(i)*power(z,2*(i-1));
                    Q1 = Q1 + q(i)*power(z,2*(i-1));
                end
                J1 = (P1*cos(fi)-z*Q1*sin(fi))*sqrt(2/pi/abs(x));
                if x<0
                    J1 = -J1;
                end
```

```
        end
        Jx = J1;
    else
        if abs(x) > n
            y = 2/x;
            bess1 = bessel(0,x);
            bess2 = bessel(1,x);
            for j=1:n-1
                Jx = j*y*bess2 - bess1;     %递推公式
                bess1 = bess2;
                bess2 = Jx;
            end
        else                        %反递推公式
            M = 2*floor(((n + floor(sqrt(40*n))))/2);
            JS = zeros(M+2 ,1);
            JS(M+2) = 0;
            JS(M+1) = 1;
            y = 2/x;
            bSum = 0;
            for k=M:-1:1
                JS(k) = JS(k+1)*y*k - JS(k+2);
            end
            BK = JS(1);
            for k=1:floor(M/2)
                BK = BK + 2*JS(2*k+1);
            end
            Jx = JS(n+1)/BK;
        end
    end
end
```

例 14-6 第一类 0 阶贝塞尔函数应用实例。计算当 $x=0,1,2,3,\cdots,10$ 时的第一类 0 阶贝塞尔函数值，并与 MATLAB 自带的函数 besselj 相比较。

解：在 MATLAB 命令行输入下列命令：

```
>> x=0:10;
>> for i=1:11
      mbsl(i) = bessel(0,x(i));
      sbsl(i) = besselj(0,x(i));
end
```

将结果列成表格比较如下：

bessel	besselj
1.000000002831410	1.000000000000000
0.765197683754860	0.765197686557970
0.223890781908570	0.223890779141240
–0.260051957719930	–0.260051954901930
–0.397149807173850	–0.397149809863850

续表

bessel	besselj
−0.177596774112340	−0.177596771314340
0.150645259491850	0.150645257251000
0.300079273614110	0.300079270519560
0.171650807172190	0.171650807137550
−0.090333611192270	−0.090333611182880
−0.245935764488980	−0.245935764451350

例 14-7 第一类 1 阶贝塞尔函数应用实例。计算当 $x = 0,1,2,3,\cdots,10$ 时的第一类 1 阶贝塞尔函数值，并与 MATLAB 自带的函数 besselj 相比较。

解：在 MATLAB 命令行输入下列命令：

```
>> x=0:10;
>> for i=1:11
     mbsl(i) = bessel(1,x(i));
     sbsl(i) = besselj(1,x(i));
end
```

将结果列成表格比较如下：

bessel	besselj
0.000000000000000	0.000000000000000
0.440050585674120	0.440050585744930
0.576724807891120	0.576724807756870
0.339058958261630	0.339058958525940
−0.066043327950040	−0.066043328023550
−0.327579138422380	−0.327579137591470
−0.276683858856970	−0.276683858127570
−0.004682825716460	−0.004682823482350
0.234636346821090	0.234636346853910
0.245311786552800	0.245311786573330
0.043472746150770	0.043472746168860

例 14-8 第一类 5 阶贝塞尔函数应用实例。计算当 $x = 1,2,3,\cdots,10$ 时的第一类 5 阶贝塞尔函数值，并与 MATLAB 自带的函数 besselj 相比较。

解：在 MATLAB 命令行输入下列命令：

```
>> x=1:10;
>> for i=1:10
     mbsl(i) = bessel(5,x(i));
     sbsl(i) = besselj(5,x(i));
end
```

将结果列成表格比较如下：

bessel	besselj
0.000249757730210	0.000249757730210
0.007039629755870	0.007039629755870
0.043028434877050	0.043028434877050
0.132086656047300	0.132086656047100
0.261140546151420	0.261140546120170
0.362087077890290	0.362087074887170
0.347896329015260	0.347896324751180
0.185774772230560	0.185774772190560
–0.055038855683040	–0.055038855669510
–0.234061528230490	–0.234061528186790

由于采用的是近似公式，因此误差是不可避免的。

14.5 第二类整数阶贝塞尔函数

第二类 n 阶（ n 为整数）贝塞尔函数的递推公式如下所示：

$$Y_{n+1}(x)=\frac{2n}{x}Y_n(x)-Y_{n-1}(x) \qquad x>0$$

其中 $Y_0(x)$ 与 $Y_1(x)$ 的计算公式如下所示：

❶ 当 $|x|<8$ 时，有：

$$Y_0(x)=\frac{A_0+A_1x^2+A_2x^4+A_3x^6+A_4x^8+A_5x^{10}}{B_0+B_1x^2+B_2x^4+B_3x^6+B_4x^8+B_5x^{10}}+\frac{2}{\pi}J_0(x)\ln x$$

$$Y_1(x)=\frac{x(C_0+C_1x^2+C_2x^4+C_3x^6+C_4x^8+C_5x^{10})}{D_0+D_1x^2+D_2x^4+D_3x^6+D_4x^8+D_5x^{10}+D_6x^{12}}+\frac{2}{\pi}[J_1(x)\ln x-\frac{1}{x}]$$

其系数的具体值如下：

$A_0=57568490574.0,\quad A_1=-13362590354.0,$
$A_2=651619640.7,\quad A_3=-11214424.18,$
$A_4=77392.33017,\quad A_5=-184.9052456$

$B_0=57568490411.0,\quad B_1=1029532985.0,$
$B_2=9494680.718,\quad B_3=59272.64853,$
$B_4=267.8532712,\quad B_5=1.0$

$C_0=72362614232.0,\quad C_1=-7895059235.0,$
$C_2=242396853.1,\quad C_3=-2972611.439,$
$C_4=15704.4826,\quad C_5=-30.16036606$

$D_0=144725228443.0,\quad D_1=2300535178.0,$
$D_2=18583304.74,\quad D_3=99447.43394,$
$D_4=376.9991397,\quad D_5=1.0$

❷ 当$|x|\geqslant 8$时，有：

$$Y_0(x)=\sqrt{\frac{2}{\pi|x|}}[R(z)\sin\theta+zS(z)\cos\theta]\ ,$$

$$Y_1(x)=\sqrt{\frac{2}{\pi|x|}}[P(z)\sin\varphi+zQ(z)\cos\varphi]\ ,$$

其中$z=\frac{8}{|x|},\theta=|x|-\frac{\pi}{4},\varphi=|x|-\frac{3\pi}{4}$，且有：

$$R(z)=r_0+r_1z^2+r_2z^4+r_3z^6+r_4z^8$$
$$S(z)=s_0+s_1z^2+s_2z^4+s_3z^6+s_4z^8$$
$$P(z)=p_0+p_1z^2+p_2z^4+p_3z^6+p_4z^8$$
$$Q(z)=q_0+q_1z^2+q_2z^4+q_3z^6+q_4z^8$$

其系数的具体值如下：

$r_0=1.0,$ $r_1=-0.001098628627,$
$r_2=0.00002734510407,$ $r_4=-0.000002073370639,$
$r_5=0.0000002093887211$
$s_0=-0.01562499995,$ $s_1=0.0001430488765,$
$s_2=-0.6911147651\times10^{-5},$ $s_3=0.7621095161\times10^{-6},$
$s_4=-0.934935152\times10^{-7}$
$p_0=1.0,$ $p_1=0.00183105,$
$p_2=-0.3516396496\times10^{-4},$ $p_3=0.2457520174\times10^{-5},$
$p_4=-0.240337019\times10^{-6}$
$q_0=0.04687499995,$ $q_1=-0.0002002690873,$
$q_2=0.8449199096\times10^{-5},$ $q_3=-0.88228987\times10^{-6},$
$q_4=0.105787412\times10^{-6}$

在 MATLAB 中编程实现的计算第二类整数阶贝塞尔函数值的函数为：bessel2

功能：用逼近法计算第二类整数阶贝塞尔函数值

调用格式：function Jx = bessel2(n,x)

其中，n：第二类贝塞尔函数的阶；

x：自变量的值；

Jx：自变量取 x 值时的第二类整数阶贝塞尔函数的值。

求第二类整数阶贝塞尔函数值的 MATLAB 程序代码如下所示：

```
function Jx = bessel2(n,x)
%第二类贝塞尔函数的阶：n
%自变量的值：x
%自变量取 x 值时的第二类整数阶贝塞尔函数的值：Jx
format long;
if n == 0        %阶数为零
```

```
    a = [-2.957821389e9;7.062834065e9;-5.123598036e8;
         1.087988129e7;-8.632792757e4;2.284622733e2];
    b = [4.0076544269e10;7.452499648e8;7.189466438e6;
         4.744726470e4;2.261030244e2;1.0];
    r =[1.0;-0.1098628627e-2;0.2734510407e-4;
        -0.2073370639e-5;0.2093887211e-6];
    s =[-0.1562499995e-1;0.1430488765e-3;-0.6911147651e-5;
        0.7621095161e-6;-0.934945152e-7];
    if abs(x) < 8        %自变量绝对值小于 8 的计算公式
        J0_u = a(1);
        J0_d = b(1);
        for i=2:6
            J0_u = J0_u + a(i)*power(x,2*(i-1));
            J0_d = J0_d + b(i)*power(x,2*(i-1));
        end
        J0 = J0_u/J0_d+2*bessel(0,x)*log(x)/pi;
    else                    %自变量绝对值不小于 8 的计算公式
        z = 8/abs(x);
        sita = abs(x) - pi/4;
        R0 = r(1);
        S0 = s(1);
        for i=2:5
            R0 = R0 + r(i)*power(z,2*(i-1));
            S0 = S0 + s(i)*power(z,2*(i-1));
        end
        J0 = (R0*sin(sita)+z*S0*cos(sita))*sqrt(2/pi/abs(x));
    end
    Jx = J0;
else
    if n == 1       %阶数为一
        c = [-4.900604943e12;1.275274390e12;-5.153438139e10;
            7.439264551e8;-4.237922726e6;8.511937935e3];
        d = [2.499580570e13;4.244419664e11;3.733650367e9;
            2.245904002e7;1.020426050e5;3.549632885e2;
            1.0];
        p = [1.0;0.183105e-2;-0.3516396496e-4;
            0.2457520174e-5;-0.240337019e-6];
        q = [0.4687499995e-1;-0.2002690873e-3;
            0.8449199096e-5;-0.88228987e-6;
            0.105787412e-6];
        if x < 8           %自变量小于 8 的计算公式
            J1_u = c(1);
            J1_d = d(1);
            for i=2:6
                J1_u = J1_u + c(i)*power(x,2*(i-1));
                J1_d = J1_d + d(i)*power(x,2*(i-1));
            end
            J1_d = J1_d + d(7)*power(x,12);
            J1 = x*J1_u/J1_d+2*(bessel(1,x)*log(x)-1/x)/pi;
        else               %自变量不小于 8 的计算公式
            z = 8/abs(x);
            fi = abs(x) - 3*pi/4;
```

```
            P1 = p(1);
            Q1 = q(1);
            for i=2:5
               P1 = P1 + p(i)*power(z,2*(i-1));
               Q1 = Q1 + q(i)*power(z,2*(i-1));
            end
            J1 = (P1*sin(fi)+z*Q1*cos(fi))*sqrt(2/pi/abs(x));
         end
         Jx = J1;
      else
         y = 2/x;
         bess1 = bessel2(0,x);
         bess2 = bessel2(1,x);
         for j=1:n-1              %递推公式
            Jx = j*y*bess2 - bess1;
            bess1 = bess2;
            bess2 = Jx;
         end
      end
   end
```

例 14-9 第二类 0 阶贝塞尔函数应用实例。计算当 $x=1,2,3,\cdots,10$ 时的第二类 0 阶贝塞尔函数值，并与 MATLAB 自带的函数 bessely 相比较。

解：在 MATLAB 命令行输入下列命令：

```
>> x=1:10;
>> for i=1:10
     mbsl(i) = bessel2(0,x(i));
     sbsl(i) = bessely(0,x(i));
end
```

将结果列成表格比较如下：

bessel	bessely
0.088256971397710	0.088256964215680
0.510375666794060	0.510375672649750
0.376850014994620	0.376850010012790
−0.016940744755700	−0.016940739325060
−0.308517623137120	−0.308517625249030
−0.288194690986530	−0.288194683981580
−0.025949755373020	−0.025949743967210
0.223521489417640	0.223521489387570
0.249936698302900	0.249936698285020
0.055671167298830	0.055671167283600

从上表的比较结果可以发现，随着 x 值的增大，两者的误差有变小的趋势。

例 14-10 第二类 1 阶贝塞尔函数应用实例。计算当 $x=1,2,3,\cdots,10$ 时的第二类 1 阶

贝塞尔函数值，并与 MATLAB 自带的函数 besselj 相比较。

解：在 MATLAB 命令行输入下列命令：

```
>> x=1:10;
>> for i=1:10
     mbsl(i) = bessel2(1,x(i));
     sbsl(i) = bessely(1,x(i));
end
```

将结果列成表格比较如下：

bessel	bessely
−0.781212467324530	−0.781212821300290
−0.106989373953970	−0.107032431540940
0.325349996686910	0.324674424791800
0.402415602299720	0.397925710557100
0.166204791116460	0.147863143391230
−0.120662531398230	−0.175010344300400
−0.175101759554330	−0.302667237024180
−0.158060461691270	−0.158060461731250
0.104314575187440	0.104314575196720
0.249015424165000	0.249015424206950

从上表的比较结果也可以发现，随着 x 值的增大，两者的误差有变小的趋势。

例 14-11 第二类 5 阶贝塞尔函数应用实例。计算当 $x=1,2,3,\cdots,10$ 时的第二类 5 阶贝塞尔函数值，并与 MATLAB 自带的函数 besselj 相比较。

解：在 MATLAB 命令行输入下列命令：

```
>> x=1:10;
>> for i=1:10
     mbsl(i) = bessel2(5,x(i));
     sbsl(i) = bessely(5,x(i));
end
```

将结果列成表格比较如下：

bessel	bessely
−260.405757124166	−260.405866625812
−9.935687619971	−9.935989128482
−1.907472261387	−1.905945953829
−0.804831204599	−0.795851421114
−0.476908010027	−0.453694822491
−0.235305653075	−0.197060888064
0.024226524554	0.063702235249

续表

bessel	bessely
0.256401065023	0.256401064990
0.285117778429	0.285117778411
0.135403047691	0.135403047689

和上面的例子一样，随着 x 值的增大，两者的误差有变小的趋势，说明当 $|x|\geqslant 8$ 时，近似公式的精度比 $|x|<8$ 时的公式更高一些。

14.6 变型的第一类整数阶贝塞尔函数

变型的第一类 n 阶（ n 为整数）贝塞尔函数的定义如下所示：

$$I_n(x)=(-1)^n J_n(ix)$$

其中 i 为虚数单位，$J_n(ix)$ 为 n 阶第一类贝塞尔函数，$I_0(x)$ 与 $I_1(x)$ 的计算公式如下所示：

❶ 当 $|x|<3.75$ 时，有：

$$I_0(x)\approx a_0+a_1y+a_2y^2+a_3y^3+a_4y^4+a_5y^5+a_6y^6$$

$$I_1(x)\approx x(c_0+c_1y+c_2y^2+c_3y^3+c_4y^4+c_5y^5+c_6y^6)$$

其中 $y=(\frac{x}{3.75})^2$，系数的具体值如下：

$$a_0=1.0,\quad a_1=3.5156229,$$
$$a_2=3.0899424,\quad a_3=1.2067492,$$
$$a_4=0.2659732,\quad a_5=0.0360768,$$
$$a_6=0.0045813$$
$$c_0=0.5,\quad c_1=0.87890594,$$
$$c_2=0.51498869,\quad c_3=0.15084934,$$
$$c_4=0.02658773,\quad c_5=0.00301532,$$
$$c_6=0.00032411$$

❷ 当 $|x|\geqslant 3.75$ 时，有：

$$I_0(x)=\frac{e^{|x|}}{\sqrt{|x|}}\sum_{k=0}^{8}b_k z^k$$

$$\begin{cases} I_1(x)=\dfrac{e^{|x|}}{\sqrt{|x|}}\displaystyle\sum_{k=0}^{8}d_k z^k, x>0 \\ I_1(-x)=-I_1(x) \end{cases}$$

其中 $z=\frac{3.75}{|x|}$，系数的具体值如下：

$$b_0=0.39894228,\quad b_1=0.01328592,$$
$$b_2=0.00225319,\quad b_3=-0.00157565,$$
$$b_4=0.00916281,\quad b_5=-0.02057706,$$
$$b_6=0.02635537,\quad b_7=-0.01647633,$$
$$b_8=0.00392377$$
$$d_0=0.39894228,\quad d_1=-0.03988024,$$
$$d_2=-0.00362018,\quad d_3=0.00163801,$$
$$d_4=-0.01031555,\quad d_5=0.02282967,$$
$$d_6=-0.02895312,\quad d_7=0.01787654,$$
$$d_8=-0.00420059$$

在计算$I_n(x)$的时候，用的是如下的逆递推关系式：

$$I_{n-1}(x)=\frac{2n}{x}I_n(x)+I_{n+1}(x),n>2$$

在 MATLAB 中编程实现计算变型的第一类整数阶贝塞尔函数值的函数为：besselm

功能：用逼近法计算变型的第一类整数阶贝塞尔函数值

调用格式：function Jx = besselm(n,x)

其中，n：变型的第一类整数阶贝塞尔函数的阶；

x：自变量的值；

Jx：自变量取 x 值时的变型的第一类整数阶贝塞尔函数的值。

求变型的第一类整数阶贝塞尔函数值的 MATLAB 程序代码如下所示：

```
function Jx = besselm(n,x)
%变型的第一类贝塞尔函数的阶：n
%自变量的值：x
%自变量取 x 值时的变型的第一类整数阶贝塞尔函数的值：Jx
format long;
if n == 0        %阶数为零
    a = [1.0;3.5156229;3.0899424;1.2067492;
         0.2659732;0.0360768;0.0045813];
    b = [0.39894228;0.01328592;0.00225319;
        -0.00157565;0.00916281;-0.02057706;
         0.02635537;-0.01647633;0.00392377];
    if abs(x) < 3.75      %自变量绝对值小于 3.75 的计算公式
        y = x^2/3.75/3.75;
        J0 = a(1);
        for i=2:7
            J0 = J0 + a(i)*power(y,(i-1));
        end
    else                         %自变量绝对值不小于 3.75 的计算公式
        z = 3.75/abs(x);
        J0 = b(1);
        for i=2:9
            J0 = J0 + b(i)*power(z,(i-1));
        end
```

```
            J0 = J0*exp(abs(x))/sqrt(abs(x));
        end
        Jx = J0;
    else
        if n == 1        %阶数为一
            a = [0.5;0.87890594;0.51498869;
                 0.15084934;0.02658773;0.00301532;0.00032411];
            b = [0.39894228;-0.03988024;-0.00362018;
                 0.00163801;-0.01031555;0.02282967;
                -0.02895312;0.01787654;,-0.00420059];
            if abs(x) < 3.75       %自变量绝对值小于 3.75 的计算公式
                y = x^2/3.75/3.75;
                J0 = a(1);
                for i=2:7
                    J0 = J0 + a(i)*power(y,(i-1));
                end
                J0 = x*J0;
            Else                    %自变量绝对值不小于 3.75 的计算公式
                z = 3.75/abs(x);
                J0 = b(1);
                for i=2:9
                    J0 = J0 + b(i)*power(z,(i-1));
                end
                J0 = J0*exp(abs(x))/sqrt(abs(x));
            end
            Jx = J0;
        else
            M = 2*floor(((n + floor(sqrt(40*n))))/2);
            JS = zeros(M+2 ,1);
            JS(M+2) = 0;
            JS(M+1) = 1;
            y = 2/x;
            for k=M:-1:1
                JS(k) = JS(k+1)*y*k + JS(k+2); %递推公式
            end
            Jx = besselm(0,x)*JS(n+1)/JS(1);
        end
    end
```

例 14-12 变型的第一类 0 阶贝塞尔函数应用实例。计算当 $x=1,2,3,\cdots,10$ 时的变型的第一类 0 阶贝塞尔函数值，并与 MATLAB 自带的函数 besseli 相比较。

解：在 MATLAB 命令行输入下列命令：

```
>> x=1:10;
>> for i=1:10
      mbsl(i) = besselm(0,x(i));
      sbsl(i) = besseli(0,x(i));
end
```

将结果列成表格比较如下：

besselm	besseli
1.266065848030	1.266065877750
2.279585307300	2.279585302340
4.880792565030	4.880792585870
11.301922170200	11.301921952140
27.239871894390	27.239871823600
67.234407236460	67.234406976480
168.593907766980	168.593908510290
427.564125313700	427.564115721800
1093.588388270760	1093.588354511370
2815.716664804150	2815.716628466260

从比较结果看，显然当 x 越来越大时，本节中的近似公式的误差也是会越来越大。

例 14-13 变型的第一类 1 阶贝塞尔函数应用实例。计算当 $x=1,2,3,\cdots,10$ 时变型的第一类 1 阶贝塞尔函数值，并与 MATLAB 自带的函数 besseli 相比较。

解：在 MATLAB 命令行输入下列命令：

```
>> x=1:10;
>> for i=1:10
     mbsl(i) = besselm(1,x(i));
     sbsl(i) = besseli(1,x(i));
end
```

将结果列成表格比较如下：

besselm	besseli
0.565159097590	0.565159103990
1.590636862500	1.590636854640
3.953370418470	3.953370217400
9.759465157390	9.759465153700
24.335641845710	24.335642142450
61.341936937310	61.341936777640
156.039096545800	156.039092869960
399.873134789590	399.873136782560
1030.914708653490	1030.914722516960
2670.988320559250	2670.988303701260

从比较结果看，变型的第一类 1 阶贝塞尔函数的近似公式的精度大概为 6 位数字。

例 14-14 变型的第一类 5 阶贝塞尔函数应用实例。计算当 $x=1,2,3,\cdots,10$ 时变型的第一类 5 阶贝塞尔函数值，并与 MATLAB 自带的函数 besseli 相比较。

解：在 MATLAB 命令行输入下列命令：

```
>> x=1:10;
>> for i=1:10
     mbsl(i) = besselm(5,x(i));
     sbsl(i) = besseli(5,x(i));
end
```

将结果列成表格比较如下：

besselm	besseli
0.000271463150	0.000271463156
0.009825679345	0.009825679323
0.091206477272	0.091206477662
0.504724372851	0.504724363113
2.157974552931	2.157974547323
7.968467773209	7.968467742396
26.885486271239	26.885486389774
85.535807176815	85.535805257921
261.479326428566	261.479318356652
777.188296432422	777.188286403260

由于当阶数大于 1 时，变型的第一类贝塞尔函数采用逆递推公式，因此随着 x 的增大，误差会越来越大。

14.7 变型的第二类整数阶贝塞尔函数

变型的第二类 n 阶（ n 为整数 ）贝塞尔函数的定义如下所示：

$$K_n(x)=\frac{\pi}{2}i^{n+1}[J_n(ix)+iY_n(ix)] \qquad x>0$$

其中 i 为虚数单位，$J_n(ix)$ 为 n 阶第一类贝塞尔函数，$Y_n(ix)$ 为 n 阶第二类贝塞尔函数。其中 $K_0(x)$ 与 $K_1(x)$ 的计算公式如下所示：

❶ 当 $|x|<2$ 时，有：

$$K_0(x)\approx\sum_{k=0}^{6}a_k y^k - I_0(x)\ln\frac{x}{2}$$

$$K_1(x)\approx\frac{1}{x}\sum_{k=0}^{6}c_k y^k + I_1(x)\ln\frac{x}{2}$$

其中 $y=(\frac{x}{2})^2$，系数的具体值如下：

$$a_0=-0.57721566,\qquad a_1=0.4227842,$$
$$a_2=0.23069756,\qquad a_3=0.0348859,$$
$$a_4=0.00262698,\qquad a_5=0.0001075,$$
$$a_6=0.0000074$$

$c_0 = 1.0$, $c_1 = 0.15443144$,
$c_2 = -0.67278579$, $c_3 = -0.18156897$,
$c_4 = -0.01919402$, $c_5 = -0.00110404$,
$c_6 = -0.00004686$

❷ 当 $|x| \geqslant 2$ 时，有：

$$K_0(x) = \frac{e^{-x}}{\sqrt{x}} \sum_{k=0}^{6} b_k z^k$$

$$K_1(x) = \frac{e^{-x}}{\sqrt{x}} \sum_{k=0}^{6} d_k z^k$$

其中 $z = \frac{2}{|x|}$，系数的具体值如下：

$b_0 = 1.25331414$, $b_1 = -0.07832358$,
$b_2 = 0.02189568$, $b_3 = -0.01062446$,
$b_4 = 0.00587872$, $b_5 = -0.0025154$,
$b_6 = 0.00053208$
$d_0 = 1.25331414$, $d_1 = 0.23498619$,
$d_2 = -0.0365562$, $d_3 = 0.01504268$,
$d_4 = -0.00780353$, $d_5 = 0.00325614$,
$d_6 = -0.00068245$

在计算 $K_n(x)$ 的时候，用的是如下的递推关系式：

$$K_{n+1}(x) = \frac{2n}{x} K_n(x) + K_{n-1}(x)$$

> 在 MATLAB 中编程实现计算变型的第二类整数阶贝塞尔函数值的函数为：besselm2
> 功能：用逼近法计算变型的第二类整数阶贝塞尔函数值
> 调用格式：function Jx = besselm2(n,x)
> 其中，n：变型的第二类整数阶贝塞尔函数的阶；
> x：自变量的值；
> Jx：自变量取 x 值时的变型的第二类整数阶贝塞尔函数的值。

求变型的第二类整数阶贝塞尔函数值的 MATLAB 程序代码如下所示：

```
function Jx = besselm2(n,x)
%变型的第二类贝塞尔函数的阶：n
%自变量的值：x
%自变量取 x 值时的变型的第二类整数阶贝塞尔函数的值：Jx
format long;
if n == 0          %阶数为零
   a = [-0.57721566;0.4227842;0.23069756;
        0.0348859;0.00262698;0.0001075;0.0000074];
   b = [1.25331414;-0.07832358;0.02189568;
      -0.01062446;0.00587872;-0.0025154;0.00053208];
   if x <= 2       %自变量小于等于 2 的计算公式
```

```
        y = x^2/4;
        J0 = a(1);
        for i=2:7
            J0 = J0 + a(i)*power(y,(i-1));
        end
        J0 = J0 + besselm(0,x)*log(2/x);
    else             %自变量大于 2 的计算公式
        z = 2/x;
        J0 = b(1);
        for i=2:7
            J0 = J0 + b(i)*power(z,(i-1));
        end
        J0 = J0*exp(-x)/sqrt(x);
    end
    Jx = J0;
else
    if n == 1      %阶数为一
        a = [1.0;0.15443144;-0.67278579;
            -0.18156897;-0.01919402;-0.00110404;-0.00004686];
        b = [1.25331414;0.23498619;-0.0365562;
             0.01504268;-0.00780353;0.00325614;-0.00068245];
        if x<=2      %自变量小于等于 2 的计算公式
            y = x^2/4;
            J0 = a(1)/x;
            for i=2:7
                J0 = J0 + a(i)*power(y,(i-1))/x;
            end
            J0 = J0 - besselm(1,x)*log(2/x);
        else      %自变量大于 2 的计算公式
            z = 2/x;
            J0 = b(1);
            for i=2:7
                J0 = J0 + b(i)*power(z,(i-1));
            end
            J0 = J0*exp(-x)/sqrt(x);
        end
        Jx = J0;
    else
        k1 = besselm2(0,x);
        k2 = besselm2(1,x);
        for j=1:n-1
            Jx = 2*j*k2/x + k1;
            k1 = k2;
            k2 = Jx;
        end
    end
end
```

例 14-15 变型的第二类 0 阶贝塞尔函数应用实例。计算当 $x=1,2,3,\cdots,10$ 时变型的第二类 0 阶贝塞尔函数值，并与 MATLAB 自带的函数 besselk 相比较。

解：在 MATLAB 命令行输入下列命令：

```
>> x=1:10;
>> for i=1:10
     mbsl(i) = besselm2(0,x(i));
     sbsl(i) = besselk(0,x(i));
end
```

将结果列成表格比较如下：

besselm2	besselk
0.421024421083420	0.421024438240710
0.113893880000000	0.113893872749530
0.034739504399300	0.034739504386280
0.011159676099470	0.011159676085850
0.003691098381960	0.003691098334040
0.001243994326710	0.001243994328010
0.000424795734120	0.000424795741870
0.000146470701180	0.000146470705220
0.000050881311530	0.000050881312960
0.000017780061930	0.000017780062320

从比较结果看，变型的第二类 0 阶贝塞尔函数的近似公式的精度约为 7 位数字。

例 14-16　变型的第二类 1 阶贝塞尔函数应用实例。计算当 $x=1,2,3,\cdots,10$ 时变型的第二类 1 阶贝塞尔函数值，并与 MATLAB 自带的函数 besselk 相比较。

解：在 MATLAB 命令行输入下列命令：

```
>> x=1:10;
>> for i=1:10
     mbsl(i) = besselm2(1,x(i));
     sbsl(i) = besselk(1,x(i));
end
```

将结果列成表格比较如下：

besselm2	besselk
0.601907231659820	0.601907230197230
0.139865880000000	0.139865881816520
0.040156431243920	0.040156431128190
0.012483498888160	0.012483498887270
0.004044613383210	0.004044613445450
0.001343919716740	0.001343919717740
0.000454182495560	0.000454182486880
0.000155369216600	0.000155369211810
0.000053637018130	0.000053637016380
0.000018648773950	0.000018648773450

从比较结果看，变型的第二类 1 阶贝塞尔函数的近似公式的精度约为 8 位数字。

例 14-17 变型的第二类 5 阶贝塞尔函数应用实例。计算当 $x=1,2,3,\cdots,10$ 时变型的第二类 5 阶贝塞尔函数值，并与 MATLAB 自带的函数 besselk 相比较。

解：在 MATLAB 命令行输入下列命令：

```
>> x=1:10;
>> for i=1:10
     mbsl(i) = besselm2(5,x(i));
     sbsl(i) = besselk(5,x(i));
end
```

将结果列成表格比较如下：

besselm2	besselk
360.960586769553	360.960589601241
9.431049240000	9.431049100596
0.937773604122	0.937773602387
0.154342548814	0.154342548726
0.032706273621	0.032706273712
0.008023718973	0.008023718980
0.002160199418	0.002160199413
0.000619358014	0.000619358011
0.000185696205	0.000185696204
0.000057541850	0.000057541850

当 $x=10$ 时，besselm2 函数和 MATLAB 的系统函数 besselk 得出的结果是一样的。

14.8 误差函数、正态分布函数

误差函数的定义为：

$$\mathrm{erf}(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\mathrm{d}t$$

误差函数可以用数值积分的各种公式来计算。

在 MATLAB 中编程实现的计算误差函数值的函数为：ErrFunc
功能：用高斯积分计算误差函数值
调用格式：function erf = ErrFunc(x)
其中，x：自变量的值；
　　erf：自变量取 x 值时的误差函数值。

求误差函数值的 MATLAB 程序代码如下所示：

```
function erf = ErrFunc(x)
```

```
%自变量的值: x
%自变量取 x 值时的误差函数值: erf
format long;
pi = 3.1415926535;
m = 1;
tol = 1;
while tol > 1.0e-6              %积分值的精度控制
    s1 = subERF(x,m);
    s2 = subERF(x,m+1);
    tol = abs(s1 - s2);
    m = m*2;
end
erf = (s1+s2)/sqrt(pi);
function tmpF = subERF(x,m)      %用高斯公式计算积分值
syms t;
h = x/m;
tmpF = 0;
for i=1:m
    tmpF = tmpF + IntGauss('exp(-t*t)',h*i-h,h*i,5);
end
```

正态分布函数的定义为:

$$P(a,\sigma,x)=\frac{1}{\sqrt{2\pi\sigma}}\int_{-\infty}^{x}\mathrm{e}^{-\frac{(t-a)^2}{2\sigma^2}}\mathrm{d}t$$

正态分布函数可以用误差函数来计算:

$$P(a,\sigma,x)=0.5+0.5\mathrm{erf}\left(\frac{x-a}{\sqrt{2}\sigma}\right)$$

例 14-18 误差函数应用实例。计算当 $x=0.1,0.5,1,2,5,10$ 时的误差函数值，并与 MATLAB 自带的函数 erf 相比较。

解: 在 MATLAB 命令行输入下列命令:

```
>> x = [ 0.1;0.5;1;2;5;10];
>> for i=1:6
     mef(i) = ErrFunc (x(i));
     sef(i) = erf(x(i));
end
```

将结果列成表格比较如下:

ErrFunc	erf
0.112462922	0.112462916
0.520499908	0.520499878
0.842700847	0.842700793
0.995322315	0.995322265
0.999999993	1
0.999999984	1

因为 ErrFunc 函数中用到了 π 的近似值，而且还有循环的精度控制，因此计算出来的结果和 MATLAB 系统函数得出的结果是有误差的。

14.9 正弦积分、余弦积分和指数积分

正弦积分的定义为：

$$Si(x)=\int_0^x \frac{\sin t}{t}\mathrm{d}t \qquad x>0$$

正弦积分的结果可以用高斯积分求得。

在 MATLAB 中编程实现的计算正弦积分值的函数为：SIx

功能：用高斯积分计算正弦积分值

调用格式：function si = SIx(x)

其中，x：自变量的值；

si：自变量取 x 值时的正弦积分值。

用高斯积分求正弦积分值的 MATLAB 程序代码如下所示。

```
function si = SIx(x)
%自变量的值：x
%自变量取 x 值时的正弦积分值：si
if x < 0
   disp('x 必须大于 0!');
   return;
end
format long;
m = 1;
tol = 1;
while tol > 1.0e-6                    %积分值的精度控制
   s1 = subSIx(x,m);
   s2 = subSIx(x,m+1);
   tol = abs(s1 - s2);
   m = m*2;
end
si = (s1+s2)/2;
function tmpS = subSIx(x,m)           %用高斯公式计算积分值
syms t;
h = x/m;
tmpS = 0;
for i=1:m
   tmpS = tmpS + IntGauss('sin(t)/t',h*i-h,h*i,5);
end
```

余弦积分的定义为：

$$Ci(x)=-\int_0^x \frac{\cos t}{t}\mathrm{d}t \qquad x>0$$

余弦积分的求法是，先把余弦积分化为

$$Ci(x)=\gamma+\ln x-\int_0^x\frac{1-\cos t}{t}\mathrm{d}t$$

再用高斯积分求 $\int_0^x\frac{1-\cos t}{t}\mathrm{d}t$，其中 γ 为欧拉常数，$\gamma=0.5772156649015328606065l$。

> 在 MATLAB 中编程实现的计算余弦积分值的函数为：CIx
> 功能：用高斯积分计算余弦积分值
> 调用格式：function ci = CIx(x)
> 其中，x：自变量的值；
> ci：自变量取 x 值时的余弦积分值。

用高斯积分求余弦积分值的 MATLAB 程序代码如下所示：

```
function ci = CIx(x)
%自变量的值：x
%自变量取 x 值时的余弦积分值：ci
function ci = CIx(x)
if x < 0
    disp('x 必须大于 0!');
    return;
end
format long;
r = 0.5772156649015328606065l;          %欧拉常数
ci = r + log(x);
m = 1;
tol = 1;
while tol > 1.0e-6
    c1 = subCIx(x,m);
    c2 = subCIx(x,m+1);
    tol = abs(c1 - c2);
    m = m*2;
end
ci = ci - (c1+c2)/2;
function tmpC = subCIx(x,m)          %用高斯公式计算积分值
syms t;
h = x/m;
tmpC = 0;
for i=1:m
    tmpC = tmpC + IntGauss('(1-cos(t))/t',h*i-h,h*i,5);
end
```

指数积分的定义为：

$$Ei(x)=-\int_x^{\infty}\frac{e^{-t}}{t}\mathrm{d}t \qquad x>0$$

计算指数积分有两种方法：

❶ 用下面的公式。

$$Ei(x) = \gamma + \ln x - \int_0^x \frac{1-e^{-t}}{t} dt$$

再用高斯积分求 $\int_0^x \frac{1-e^{-t}}{t} dt$ ，其中 γ 为欧拉常数。

在 MATLAB 中编程实现的计算指数积分值的函数为：EIx

功能：用高斯积分计算指数积分值

调用格式：function ei = EIx (x)

其中，x：自变量的值；

ei：自变量取 x 值时的指数积分值

用高斯积分求指数积分值的 MATLAB 程序代码如下所示：

```
function ei = EIx(x)
%自变量的值：x
%自变量取 x 值时的指数积分值：ei
if x < 0
    disp('x 必须大于 0!');
    return;
end
format long;
r = 0.57721566490153286060651;
ei = r + log(x);
m = 1;
tol = 1;
while tol > 1.0e-6
    e1 = subEIx(x,m);
    e2 = subEIx(x,m+1);
    tol = abs(e1 - e2);
    m = m*2;
end
ei = ei - (e1+e2)/2;
function tmpE = subEIx(x,m)          %用高斯公式计算积分值
syms t;
h = x/m;
tmpE = 0;
for i=1:m
    tmpE = tmpE + IntGauss('(1-exp(-t))/t',h*i-h,h*i,5);
end
```

❷ 用逼近法。

当 $0<x<1$ 时，

$$Ei(x) \approx \ln x - (e_1 + e_2 x + \cdots + e_6 x^5)$$

其中的系数如下：

$$e_1 = -0.57721566, \quad e_2 = 0.99999193$$
$$e_3 = -0.24991055, \quad e_4 = 0.05519968$$
$$e_5 = -0.00976004, \quad e_6 = 0.00107857$$

当 $x \geqslant 1$ 时，

$$Ei(x) \approx -\frac{e^{-x}}{x}\left(\frac{r_1 + r_2 x + r_3 x^2 + r_4 x^3 + x^4}{s_1 + s_2 x + s_3 x^2 + s_4 x^3 + x^4}\right)$$

其中的系数如下：

$$r_1 = 0.2677737343, \quad r_2 = 8.6347608925$$
$$r_3 = 18.059016973, \quad r_4 = 8.5733287401$$
$$s_1 = 3.9584969228, \quad s_2 = 21.0996530827$$
$$s_3 = 25.6329561486, \quad s_4 = 9.5733223454$$

在 MATLAB 中编程实现的计算指数积分值的函数为：EIx2

功能：用逼近法计算指数积分值

调用格式：function ei = EIx2 (x)

其中，x：自变量的值；

ei：自变量取 x 值时的指数积分值。

求指数积分值的 MATLAB 程序代码如下所示：

```
function ei = EIx2(x)
%自变量的值：x
%自变量取 x 值时的指数积分值：ei
if x < 0
   disp('x 必须大于 0!');
   return;
end
format long;
cof = [-0.57721566;0.99999193;-0.24991055;
       0.05519968;-0.0097004;0.00107857];
cr = [0.2677737343;8.6347608925;18.059016973;
      8.5733287401;1.0];
cs = [3.9584969228;21.0996530827;25.6329561486;
      9.5733223454;1.0];
if x < 1      %自变量小于 1 时的计算公式
   ei = log(x);
   for i=1:6
      ei = ei - cof(i)*power(x,i-1);
   end
else          %自变量不小于 1 时的计算公式
   ei = -exp(-x)/x;
   ER = 0;
   ES = 0;
   for i=1:5
      ER = ER + cr(i)*power(x,i-1);
      ES = ES + cs(i)*power(x,i-1);
   end
   ei = ei*ER/ES;
end
```

例 14-19 正弦积分函数应用实例。计算当 $x=10$ 时的正弦积分值。

解：在 MATLAB 命令行输入下列命令：

```
>>  msin = SIx(10)
msin = 1.65834776232614
```

用 MATLAB 自带的正弦积分函数 sinint 求得的结果如下所示：

```
>> msin = sinint(10)
msin = 1.65834759421887
```

例 14-20 余弦积分函数应用实例。计算当 $x=10$ 时的余弦积分值。

解：在 MATLAB 命令行输入下列命令：

```
>> mcos = CIx(10)
mcos = -0.04545666829043
```

用 MATLAB 自带的余弦积分函数 cosint 求得的结果如下：

```
>> mcos = cosint(10)
mcos = -0.04545643300446
```

例 14-21 指数积分应用实例。用两种方法计算当 $x=2$ 时的指数积分值。

解：在 MATLAB 命令行输入下列命令：

```
>> mexp = EIx(2)
mexp = -0.04890055114497
>> mexp = EIx2(2)
mexp = -0.04890050998053
```

两种方法算出来的结果稍微有点误差。

14.10 第一类椭圆积分

第一类椭圆积分的定义为：

$$F(k,\varphi)=\int_0^{\varphi}\frac{1}{\sqrt{1-k^2\sin^2\theta}}\mathrm{d}\theta \quad 0\leqslant k\leqslant 1,\varphi\text{任意}$$

当 $|\varphi|>\frac{\pi}{2}$ 时，有如下关系式：

$$F(k,n\pi\pm\varphi)=2nF(k,\frac{\pi}{2})\pm F(k,\varphi)$$

第一类椭圆积分可以用高斯积分公式来求。

在 MATLAB 中编程实现的计算第一类椭圆积分值的函数为：Ellipint1

功能：用高斯积分计算第一类椭圆积分值

调用格式：y = Ellipint1(x,k)

其中，x：自变量的值；

k：积分参数；
y：自变量取 x 值时的第一类椭圆积分值。

求第一类椭圆积分的 MATLAB 程序代码如下所示：

```
function y = Ellipint1(x,k)
%自变量的值：x
%积分参数：k
%自变量取 x 值时的第一类椭圆积分值：y
format long;
m = 2;
tol = 1;
while tol > 1.0e-6     %积分值的精度控制
    y1 = subEllip(x,k,m);
    y2 = subEllip(x,k,m*2);
    tol = abs(y1 - y2);
    m = m*2;
end
y = (y1+y2)/2;
function tmpS = subEllip(x,k,m)     %用高斯公式计算积分值
syms t;
h = x/m;
tmpS = 0;
u = k*k;
for i=1:m
    f = 1/sqrt(1-u*sin(t)*sin(t));
    tmpS = tmpS + IntGauss(f,h*i-h,h*i,5);
end
```

例 14-22 第一类椭圆积分函数应用实例。计算当 $x=1.5, k=0.8$ 时的第一类椭圆积分值。

解：在 MATLAB 命令行输入下列命令：

```
>> melli = Ellipint1(1.5,0.8)
melli = 1.87748339101508
```

14.11 第二类椭圆积分

第二类椭圆积分的定义为：

$$E(k,\varphi)=\int_0^{\varphi}\sqrt{1-k^2\sin^2\theta}\mathrm{d}\theta \qquad 0\leqslant k\leqslant 1, \varphi\text{任意}$$

当 $|\varphi|>\dfrac{\pi}{2}$ 时，有如下关系式：

$$E(k,n\pi\pm\varphi)=2nE(k,\frac{\pi}{2})\pm E(k,\varphi)$$

第二类椭圆积分可以用高斯积分公式来求。

在 MATLAB 中编程实现的计算第二类椭圆积分值的函数为：Ellipint2

功能：用高斯积分计算第二类椭圆积分值

调用格式：y = Ellipint2(x,k)

其中，x：自变量的值；

k：积分参数；

y：自变量取 x 值时的第二类椭圆积分值。

求第二类椭圆积分的 MATLAB 程序代码如下所示：

```
function y = Ellipint2(x,k)
%自变量的值：x
%积分参数：k
%自变量取 x 值时的第一类椭圆积分值：y
format long;
m = 2;
tol = 1;
while tol > 1.0e-6    %积分值的精度控制
    y1 = subEllip(x,k,m);
    y2 = subEllip(x,k,m*2);
    tol = abs(y1 - y2);
    m = m*2;
end
y = (y1+y2)/2;
function tmpS = subEllip(x,k,m)  %用高斯公式计算积分值
syms t;
h = x/m;
tmpS = 0;
u = k*k;
for i=1:m
    f = (1-u*sin(t)*sin(t));
    tmpS = tmpS + IntGauss(f,h*i-h,h*i,5);
end
```

例 14-23 第二类椭圆积分函数应用实例。计算当 $x=1.5,k=0.8$ 时的第二类椭圆积分值。

解：在 MATLAB 命令行输入下列命令：

```
>> melli = Ellipint2(1.5,0.8)
melli = 1.04257925436752
```

14.12 小结

本章介绍了数理方程中常用的几种特殊函数的计算方法，基本上都是基于展开近似的方法进行计算，其结果的精度当然和展开的项数和常数的计算精度有很大关系。MATLAB 在特殊函数的计算方面采用了较多的有效位数，因此其精度很高，而本章的例题结果都以 MATLAB 算出的结果为基准进行比较。

第 15 章　常微分方程的初值问题

根据给定的初始条件，确定常微分方程唯一解的问题叫常微分方程初值问题。大多数实际的常微分方程往往结构非常复杂，要给出一般方程解的表达式非常困难。

利用计算机作为辅助计算工具，并根据高等数学的有关知识将欧拉公式、改进的欧拉公式、经典龙格-库塔方法编成 MATLAB 程序算法，充分利用了计算机的速度优势，大大减轻了工程技术人员的劳动强度，同时也使计算结果更加可靠、准确。

15.1 欧拉法

微分方程里最简单的方程形式莫过于一阶常微分方程的初值问题了，即

$$\begin{cases} \dfrac{dy}{dx} = f(x,y) \quad a \leqslant x \leqslant b \\ y(a) = y_0 \end{cases}$$

其中 a,b 为常数。

因为其简单但又是求解其他方程的基础，所以发展了许多典型的解法，下面介绍的数值解法都是针对于上式进行求解，所有算法中的 f 就代表上式中的 $f(x,y)$，而 f_y 表示 $\dfrac{\partial f(x,y)}{\partial y}$，$f_x$ 表示 $\dfrac{\partial f(x,y)}{\partial x}$。

欧拉法（Euler）是简单有效的常微分方程数值解法，欧拉法有多种形式的算法，常见的有简单欧拉法、隐式欧拉法和改进的欧拉法，下面分别进行讲述。

15.1.1 简单欧拉法

简单欧拉法是一种单步递推算法。简单欧拉法的公式如下所示：

$$y_{n+1} = y_n + hf(x_n, y_n)$$

简单欧拉法的算法过程介绍如下。

❶ 给出自变量 x 的定义域 $[a,b]$，初始值 y_0 及步长 h。

❷ 对 $k = 0,1,\cdots,(b-a)/h$，计算

$$y_{k+1} = y_k + hf(x_k, y_k)$$

算法结束。

> 在 MATLAB 中编程实现的简单欧拉法的函数为：DEEuler
>
> 功能：用简单欧拉法求一阶常微分方程的数值解

调用格式：y = DEEuler(f, h,a,b,y0,varvec)
其中，f：一阶常微分方程的一般表达式的右端函数；
h：积分步长；
a：自变量取值下限；
b：自变量取值上限；
y0：函数初值；
varvec：常微分方程的变量组。

简单欧拉法的 MATLAB 程序代码如下所示：

```
 function y = DEEuler(f, h,a,b,y0,varvec)
%一阶常微分方程的一般表达式的右端函数：f
%积分步长：h
%自变量取值下限：a
%自变量取值上限：b
%函数初值：y0
%积分步长：h
%常微分方程的变量组：varvec
format long;
N = (b-a)/h;
y = zeros(N+1,1);
x = a:h:b;
y(1) = y0;
for i=2:N+1
   y(i) = y(i-1)+h*Funval(f,varvec,[x(i-1), y(i-1)]);  %简单欧拉法的迭代公式
end
format short;
```

例 15-1 简单欧拉法求解一阶常微分方程应用实例。用简单欧拉法求下面常微分方程的数值解。

$$\begin{cases} \dfrac{\mathrm{d}y}{\mathrm{d}x} = y & 0 \leqslant x \leqslant 1 \\ y(0) = 1 \end{cases}$$

解： 在 MATLAB 命令窗口中输入：

```
>> syms u v;
>> z=v;
>> y = DEEuler(z, 0.1,0,1,1,[u v])
y =  1.0000
     1.1000
     1.2100
     1.3310
     1.4641
     1.6105
     1.7716
     1.9487
     2.1436
```

```
    2.3579
    2.5937
```

通过简单计算，可得出常微分方程

$$\begin{cases} \dfrac{dy}{dx} = y & 0 \leqslant x \leqslant 1 \\ y(0) = 1 \end{cases}$$

的解为 $y = e^x$，将数值解与理论解列成表格如下：

x	数值解	理论解
0	1.0000	1.0000
0.1	1.1000	1.1052
0.2	1.2100	1.2214
0.3	1.3310	1.3499
0.4	1.4641	1.4918
0.5	1.6105	1.6487
0.6	1.7716	1.8221
0.7	1.9487	2.0138
0.8	2.1436	2.2255
0.9	2.3579	2.4596
1.0	2.5937	2.7183

从比较的结果来看，x 的值越大，误差也越大。

15.1.2 隐式欧拉法

隐式欧拉法也叫后退欧拉法，隐式欧拉法的公式如下所示：

$$y_{n+1} = y_n + hf(x_{n+1}, y_{n+1})$$

隐式欧拉法是一阶精度的方法，比它精度高的公式是：

$$y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y_{n+1})]$$

隐式欧拉法的算法过程介绍如下。

❶ 给出自变量 x 的定义域 $[a,b]$，初始值 y_0 及步长 h。

❷ 对 $k = 0,1,\cdots,(b-a)/h$，用牛顿法或其他方法求解方程

$$y_{k+1} = y_k + hf(x_{k+1}, y_{k+1})$$

得出 y_{k+1}。

算法结束。

在 MATLAB 中编程实现的隐式欧拉法的函数为：DEimpEuler

功能：用隐式欧拉法求一阶常微分方程的数值解

调用格式：y = DEimpEuler(f, h,a,b,y0,varvec)

其中，f：一阶常微分方程的一般表达式的右端函数；

h：积分步长；
a：自变量取值下限；
b：自变量取值上限；
y0：函数初值；
varvec：常微分方程的变量组。

隐式欧拉法的 MATLAB 程序代码如下所示：

```
function y = DEimpEuler(f, h,a,b,y0,varvec)
%一阶常微分方程的一般表达式的右端函数：f
%积分步长：h
%自变量取值下限：a
%自变量取值上限：b
%函数初值：y0
%积分步长：h
%常微分方程的变量组：varvec
format long;
N = (b-a)/h;
y = zeros(N+1,1);
y(1) = y0;
x = a:h:b;
var = findsym(f);
for i=2:N+1
   fx = Funval(f,var(1),x(i));
   gx = y(i-1)+h*fx - varvec(2);
   y(i) = NewtonRoot(gx,-10,10,eps); %用牛顿法得出下步的迭代值
end
format short;
```

例 15-2 隐式欧拉法求解一阶常微分方程应用实例。用隐式欧拉法求下面常微分方程的数值解。

$$\begin{cases}\dfrac{\mathrm{d}y}{\mathrm{d}x}=y & 0\leqslant x\leqslant 1\\ y(0)=1\end{cases}$$

解： 在 MATLAB 命令窗口中输入：

```
>> syms u v;
>> z=v;
>> y = DEimpEuler(z, 0.1,0,1,1,[u v])
y =  1.0000
     1.1111
     1.2346
     1.3717
     1.5242
     1.6935
     1.8817
     2.0908
```

```
2.3231
2.5812
2.8680
```

隐式欧拉法的结果比欧拉法稍好一点，但是随着 x 的增大，误差也是越来越大。

15.1.3 改进的欧拉法

改进的欧拉法是一种二阶显式求解法，其计算公式如下所示：

$$\begin{cases} t = y_n + hf(x_n, y_n) \\ y_{n+1} = y_n + \dfrac{h}{2}[f(x_n, y_n) + f(x_{n+1}, t)] \end{cases}$$

在 MATLAB 中编程实现改进的欧拉法的函数为：DEModifEuler

功能：用改进的欧拉法求一阶常微分方程的数值解

调用格式：y = DEModifEuler (f, h,a,b,y0,varvec)

其中，f：一阶常微分方程的一般表达式的右端函数；

h：积分步长；

a：自变量取值下限；

b：自变量取值上限；

y0：函数初值；

varvec：常微分方程的变量组。

改进的欧拉法的 MATLAB 程序代码如下所示：

```
function y = DEModifEuler (f, h,a,b,y0,varvec)
%一阶常微分方程的一般表达式的右端函数：f
%积分步长：h
%自变量取值下限：a
%自变量取值上限：b
%函数初值：y0
%积分步长：h
%常微分方程的变量组：varvec
format long;
N = (b-a)/h;
y = zeros(N+1,1);
y(1) = y0;
x = a:h:b;
var = findsym(f);
for i=2:N+1
       v1 = Funval(f,varvec,[x(i-1) y(i-1)]);
       t = y(i-1) + h*v1;    %迭代公式的第一步
       v2 = Funval(f,varvec,[x(i) t]);
       y(i) = y(i-1)+h*(v1+v2)/2;  %迭代公式的第二步
end
format short;
```

例 15-3 改进的欧拉法求解一阶常微分方程应用实例。用改进的欧拉法求下面常微分方程的数值解。

$$\begin{cases} \dfrac{dy}{dx} = y \quad 0 \leqslant x \leqslant 1 \\ y(0) = 1 \end{cases}$$

解： 在 MATLAB 命令窗口中输入：

```
>> syms u v;
>> z=v;
>> y = DEModifEuler(z, 0.1,0,1,1,[u v])
y =  1.0000
     1.1050
     1.2210
     1.3492
     1.4909
     1.6474
     1.8204
     2.0116
     2.2228
     2.4562
     2.7141
```

将改进的欧拉法的结果与理论值比较如下：

x	数值解	理论解
0	1.0000	1.0000
0.1	1.1050	1.1052
0.2	1.2210	1.2214
0.3	1.3492	1.3499
0.4	1.4909	1.4918
0.5	1.6474	1.6487
0.6	1.8204	1.8221
0.7	2.0116	2.0138
0.8	2.2228	2.2255
0.9	2.45629	2.4596
1.0	2.7141	2.7183

从上表可以看出，改进的欧拉法的结果十分精确。

15.2 龙格-库塔法

龙格-库塔法有显式和隐式之分，R 级的显式龙格-库塔法的形式为：

$$y_{n+1} = y_n + h\sum_{r=1}^{R} c_r K_r$$

其中

$$\begin{cases} K_1 = f(x_n, y_n) \\ K_r = f(x_n + p_r h, y_n + \sum_{m=1}^{r-1} q_{rm} K_m) \end{cases} \quad (r = 2,3,\cdots,R)$$

15.2.1 二阶龙格-库塔法

二阶龙格-库塔法有多种形式，除了改进的欧拉法外，还有中点法和休恩法。

- 中点法的计算公式为：

$$y_{n+1} = y_n + hf\left[x_n + \frac{h}{2}, y_n + \frac{h}{2} f(x_n, y_n)\right]$$

> 在 MATLAB 中编程实现的中点法的函数为：DELGKT2_mid
>
> 功能：用中点法求一阶常微分方程的数值解
>
> 调用格式：y = DELGKT2_mid (f, h,a,b,y0,varvec)
>
> 其中，f：一阶常微分方程的一般表达式的右端函数；
>
> h：积分步长；
>
> a：自变量取值下限；
>
> b：自变量取值上限；
>
> y0：函数初值；
>
> varvec：常微分方程的变量组。

中点法的 MATLAB 程序代码如下所示：

```
 function y = DELGKT2_mid (f, h,a,b,y0,varvec)
%一阶常微分方程的一般表达式的右端函数：f
%积分步长：h
%自变量取值下限：a
%自变量取值上限：b
%函数初值：y0
%积分步长：h
%常微分方程的变量组：varvec
format long;
N = (b-a)/h;
y = zeros(N+1,1);
y(1) = y0;
x = a:h:b;
var = findsym(f);
for i=2:N+1
    v1 = Funval(f,varvec,[x(i-1) y(i-1)]);
    t = y(i-1) + h*v1/2;
    v2 = Funval(f,varvec,[x(i)+h/2 t]);
    y(i) = y(i-1)+h*v2;   %中点法的迭代公式
```

```
end
format short;
```

- 休恩法的计算公式为：

$$\begin{cases} K_1 = f(x_n, y_n) \\ K_2 = f(x_n + \dfrac{2h}{3}, y_n + \dfrac{2h}{3}K_1) \\ y_{n+1} = y_n + \dfrac{h}{4}(K_1 + 3K_2) \end{cases}$$

在 MATLAB 中编程实现的休恩法的函数为：DELGKT2_suen

功能：用休恩法求一阶常微分方程的数值解

调用格式：y = DELGKT2_suen (f, h,a,b,y0,varvec)

其中，f：一阶常微分方程的一般表达式的右端函数；

h：积分步长；

a：自变量取值下限；

b：自变量取值上限；

y0：函数初值；

varvec：常微分方程的变量组。

休恩法的 MATLAB 程序代码如下所示：

```
function y = DELGKT2_suen (f, h,a,b,y0,varvec)
%一阶常微分方程的一般表达式的右端函数：f
%积分步长：h
%自变量取值下限：a
%自变量取值上限：b
%函数初值：y0
%积分步长：h
%常微分方程的变量组：varvec
format long;
N = (b-a)/h;
y = zeros(N+1,1);
y(1) = y0;
x = a:h:b;
var = findsym(f);
for i=2:N+1
    K1 = Funval(f,varvec,[x(i-1) y(i-1)]);   %休恩公式的第一步
    K2 = Funval(f,varvec,[x(i-1)+2*h/3 y(i-1)+K1*2*h/3]);  %休恩公式的第二步
    y(i) = y(i-1)+h*(K1+3*K2)/4;  %休恩公式的第三步
end
format short;
```

例 15-4 二阶龙格-库塔法求解一阶常微分方程应用实例。用二阶龙格-库塔法求下面常微分方程的数值解。

$$\begin{cases}\dfrac{\mathrm{d}y}{\mathrm{d}x}=x-y+1 & 0\leqslant x\leqslant 1\\ y(0)=1\end{cases}$$

解： 在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = x-y+1;
>> yy = DELGKT2_mid(z,0.1,0,1,1,[x y])
yy =  1.0000
      1.0150
      1.0381
      1.0685
      1.1055
      1.1484
      1.1968
      1.2501
      1.3079
      1.3696
      1.4350
>> yy = DELGKT2_suen(z,0.1,0,1,1,[x y])
yy =  1.0000
      1.0050
      1.0190
      1.0412
      1.0708
      1.1071
      1.1494
      1.1972
      1.2500
      1.3072
      1.3685
```

通过简单计算，可得出常微分方程

$$\begin{cases}\dfrac{\mathrm{d}y}{\mathrm{d}x}=x-y+1 & 0\leqslant x\leqslant 1\\ y(0)=1\end{cases}$$

的解为 $y=\mathrm{e}^{-x}+x$，将数值解与理论解列成表格如下：

x	中点法	休恩法	理论解
0	1.0000	1.0000	1.0000
0.1	1.0150	1.0050	1.0048
0.2	1.0381	1.0190	1.0187
0.3	1.0685	1.0412	1.0408
0.4	1.1055	1.0708	1.0703
0.5	1.1484	1.1071	1.1065
0.6	1.1968	1.1494	1.1488
0.7	1.2501	1.1972	1.1966

续表

x	中点法	休恩法	理论解
0.8	1.3079	1.2500	1.2493
0.9	1.3696	1.3072	1.3066
1.0	1.4350	1.3685	1.3679

可见休恩法要比中点法好。

15.2.2 三阶龙格-库塔法

三阶龙格-库塔法也有多种形式。

- 休恩三阶法的计算公式为:

$$\begin{cases} K_1 = f(x_n, y_n) \\ K_2 = f(x_n + \frac{h}{3}, y_n + \frac{h}{3}K_1) \\ K_3 = f(x_n + \frac{2h}{3}, y_n + \frac{2h}{3}K_2) \\ y_{n+1} = y_n + \frac{h}{4}(K_1 + 3K_3) \end{cases}$$

在 MATLAB 中编程实现的休恩三阶法的函数为: DELGKT3_suen

功能: 用休恩三阶法求一阶常微分方程的数值解

调用格式: y = DELGKT3_suen (f, h,a,b,y0,varvec)

其中, f: 一阶常微分方程的一般表达式的右端函数;

h: 积分步长;

a: 自变量取值下限;

b: 自变量取值上限;

y0: 函数初值;

varvec: 常微分方程的变量组。

休恩三阶法的 MATLAB 程序代码如下所示:

```
 function y = DELGKT3_suen (f, h,a,b,y0,varvec)
%一阶常微分方程的一般表达式的右端函数: f
%积分步长: h
%自变量取值下限: a
%自变量取值上限: b
%函数初值: y0
%积分步长: h
%常微分方程的变量组: varvec
format long;
N = (b-a)/h;
y = zeros(N+1,1);
y(1) = y0;
```

```
x = a:h:b;
var = findsym(f);
for i=2:N+1
    K1 = Funval(f,varvec,[x(i-1) y(i-1)]);  %休恩三阶公式的第一步
    K2 = Funval(f,varvec,[x(i-1)+h/3 y(i-1)+K1*h/3]);  %休恩三阶公式的第二步
    K3 = Funval(f,varvec,[x(i-1)+2*h/3 y(i-1)+K2*2*h/3]); %休恩三阶公式的第三步
    y(i) = y(i-1)+h*(K1+3*K3)/4;  %休恩三阶公式的第四步
end
format short;
```

- 库塔三阶法的计算公式为：

$$\begin{cases} K_1 = f(x_n, y_n) \\ K_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}K_1) \\ K_3 = f(x_n + h, y_n - hK_1 + 2hK_2) \\ y_{n+1} = y_n + \frac{h}{6}(K_1 + 4K_2 + K_3) \end{cases}$$

在 MATLAB 中编程实现的库塔三阶法的函数为：DELGKT3_kuta

功能：用库塔三阶法求一阶常微分方程的数值解

调用格式：y = DELGKT3_kuta (f, h,a,b,y0,varvec)

其中，f：一阶常微分方程的一般表达式的右端函数；

h：积分步长；

a：自变量取值下限；

b：自变量取值上限；

y0：函数初值；

varvec：常微分方程的变量组。

库塔三阶法的 MATLAB 程序代码如下所示：

```
 function y = DELGKT3_kuta (f, h,a,b,y0,varvec)
%一阶常微分方程的一般表达式的右端函数：f
%积分步长：h
%自变量取值下限：a
%自变量取值上限：b
%函数初值：y0
%积分步长：h
%常微分方程的变量组：varvec
format long;
N = (b-a)/h;
y = zeros(N+1,1);
y(1) = y0;
x = a:h:b;
var = findsym(f);
for i=2:N+1
   K1 = Funval(f,varvec,[x(i-1) y(i-1)]);   %库塔三阶公式的第一步
```

```
    K2 = Funval(f,varvec,[x(i-1)+h/2 y(i-1)+K1*h/2]); %库塔三阶公式的第二步
    K3 = Funval(f,varvec,[x(i-1)+h y(i-1)-h*K1+K2*2*h]); %库塔三阶公式的第三步
    y(i) = y(i-1)+h*(K1+4*K2+K3)/6;  %库塔三阶公式的第四步
end
format short;
```

例 15-5 三阶龙格-库塔法求解一阶常微分方程应用实例。用三阶龙格-库塔法求下面常微分方程的数值解。

$$\begin{cases} \dfrac{dy}{dx} = x - y + 1 \quad 0 \leqslant x \leqslant 1 \\ y(0) = 1 \end{cases}$$

解： 在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = x-y+1;
>> yy = DELGKT3_suen(z,0.1,0,1,1,[x y])
yy =  1.0000
      1.0048
      1.0187
      1.0408
      1.0703
      1.1065
      1.1488
      1.1966
      1.2493
      1.3066
      1.3679
>> yy = DELGKT3_kuta(z,0.1,0,1,1,[x y])
yy =  1.0000
      1.0048
      1.0187
      1.0408
      1.0703
      1.1065
      1.1488
      1.1966
      1.2493
      1.3066
      1.3679
```

两种方法的精度几乎一样。

15.2.3 四阶龙格-库塔法

四阶龙格-库塔法有三种形式。

- 经典龙格-库塔法的计算公式为：

$$\begin{cases} K_1 = f(x_n, y_n) \\ K_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}K_1) \\ K_3 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}K_2) \\ K_4 = f(x_n + h, y_n + hK_3) \\ y_{n+1} = y_n + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4) \end{cases}$$

在 MATLAB 中编程实现的经典龙格-库塔法的函数为：DELGKT4_lungkuta

功能：用经典龙格-库塔法求一阶常微分方程的数值解

调用格式：y = DELGKT4_lungkuta (f, h,a,b,y0,varvec)

其中，f：一阶常微分方程的一般表达式的右端函数；

h：积分步长；

a：自变量取值下限；

b：自变量取值上限；

y0：函数初值；

varvec：常微分方程的变量组。

经典龙格-库塔法的 MATLAB 程序代码如下所示：

```
function y = DELGKT4_lungkuta (f, h,a,b,y0,varvec)
%一阶常微分方程的一般表达式的右端函数：f
%积分步长：h
%自变量取值下限：a
%自变量取值上限：b
%函数初值：y0
%积分步长：h
%常微分方程的变量组：varvec
format long;
N = (b-a)/h;
y = zeros(N+1,1);
y(1) = y0;
x = a:h:b;
var = findsym(f);
for i=2:N+1
    K1 = Funval(f,varvec,[x(i-1) y(i-1)]); %经典龙格-库塔三阶公式的第一步
    K2 = Funval(f,varvec,[x(i-1)+h/2 y(i-1)+K1*h/2]); %经典龙格-库塔三阶公式的第二步
    K3 = Funval(f,varvec,[x(i-1)+h/2 y(i-1)+K2*h/2]); %经典龙格-库塔三阶公式的第三步
    K4 = Funval(f,varvec,[x(i-1)+h y(i-1)+h*K3]); %经典龙格-库塔三阶公式的第四步
    y(i) = y(i-1)+h*(K1+2*K2+2*K3+K4)/6; %经典龙格-库塔三阶公式的第五步
end
format short;
```

- 基尔法的计算公式为：

$$
\begin{cases}
K_1 = f(x_n, y_n) \\
K_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}K_1) \\
K_3 = f(x_n + \frac{h}{2}, y_n + \frac{(\sqrt{2}-1)h}{2}K_1 + \frac{(2-\sqrt{2})h}{2}K_2) \\
K_4 = f(x_n + h, y_n - \frac{\sqrt{2}h}{2}K_2 + \frac{(2+\sqrt{2})h}{2}K_3) \\
y_{n+1} = y_n + \frac{h}{6}(K_1 + (2-\sqrt{2})K_2 + (2+\sqrt{2})K_3 + K_4)
\end{cases}
$$

在 MATLAB 中编程实现的基尔法的函数为：DELGKT4_jer

功能：用基尔法求一阶常微分方程的数值解

调用格式：y = DELGKT4_jer (f, h,a,b,y0,varvec)

其中，f：一阶常微分方程的一般表达式的右端函数；

h：积分步长；

a：自变量取值下限；

b：自变量取值上限；

y0：函数初值；

varvec：常微分方程的变量组。

基尔法的 MATLAB 程序代码如下所示：

```
function y = DELGKT4_jer (f, h,a,b,y0,varvec)
%一阶常微分方程的一般表达式的右端函数：f
%积分步长：h
%自变量取值下限：a
%自变量取值上限：b
%函数初值：y0
%积分步长：h
%常微分方程的变量组：varvec
format long;
N = (b-a)/h;
C2 = sqrt(2);
y = zeros(N+1,1);
y(1) = y0;
x = a:h:b;
var = findsym(f);
for i=2:N+1
    K1 = Funval(f,varvec,[x(i-1) y(i-1)]); %基尔公式的第一步
    K2 = Funval(f,varvec,[x(i-1)+h/2 y(i-1)+K1*h/2]);  %基尔公式的第二步
    K3 = Funval(f,varvec,[x(i-1)+h/2 y(i-1)+K1*h*(C2-1)/2+K2*h*(2-C2)/2];
    %基尔公式的第三步
    K4 = Funval(f,varvec,[x(i-1)+h y(i-1)-K2*h*C2/2+K3*h*(2+C2)/2]);
    %基尔公式的第四步
    y(i) = y(i-1)+h*(K1+(2-C2)*K2+(2+C2)*K3+K4)/6; %基尔公式的第五步
end
```

```
format short;
```

- 变型龙格-库塔法的计算公式为：

$$\begin{cases} K_1 = f(x_n, y_n) \\ K_2 = f(x_n + \dfrac{h}{3}, y_n + \dfrac{h}{3}K_1) \\ K_3 = f[x_n + \dfrac{2h}{3}, y_n - \dfrac{h}{3}(K_1 - 3K_2)] \\ K_4 = f[x_n + h, y_n + h(K_1 - K_2 + K_3)] \\ y_{n+1} = y_n + \dfrac{h}{8}(K_1 + 3K_2 + 3K_3 + K_4) \end{cases}$$

在 MATLAB 中编程实现的变型龙格-库塔法的函数为：DELGKT4_qt

功能：用变型龙格-库塔法求一阶常微分方程的数值解

调用格式：y = DELGKT4_qt (f, h,a,b,y0,varvec)

其中，f：一阶常微分方程的一般表达式的右端函数；

h：积分步长；

a：自变量取值下限；

b：自变量取值上限；

y0：函数初值；

varvec：常微分方程的变量组。

变型龙格-库塔法的 MATLAB 程序代码如下所示：

```
function y = DELGKT4_qt (f, h,a,b,y0,varvec)
%一阶常微分方程的一般表达式的右端函数：f
%积分步长：h
%自变量取值下限：a
%自变量取值上限：b
%函数初值：y0
%积分步长：h
%常微分方程的变量组：varvec
format long;
N = (b-a)/h;
y = zeros(N+1,1);
y(1) = y0;
x = a:h:b;
var = findsym(f);
for i=2:N+1
   K1 = Funval(f,varvec,[x(i-1) y(i-1)]); %变型龙格–库塔公式的第一步
   K2 = Funval(f,varvec,[x(i-1)+h/3 y(i-1)+K1*h/3]); %变型龙格–库塔公式的第二步
   K3 = Funval(f,varvec,[x(i-1)+2*h/3 y(i-1)-K1*h/3+K2*h]);
   %变型龙格–库塔公式的第三步
   K4 = Funval(f,varvec,[x(i-1)+h y(i-1)+h*(K1-K2+K3)]);
   %变型龙格–库塔公式的第四步
   y(i) = y(i-1)+h*(K1+3*K2+3*K3+K4)/8; %变型龙格–库塔公式的第五步
end
```

```
format short;
```

例 15-6　四阶龙格-库塔法求解一阶常微分方程应用实例。用四阶龙格-库塔法求下面常微分方程的数值解。

$$\begin{cases} \dfrac{dy}{dx} = 1 + \ln(x+1) \quad 0 \leqslant x \leqslant 1 \\ y(0) = 1 \end{cases}$$

解： 在 MATLAB 命令窗口中输入：

```
>> z =1+log(x+1);
>> yy = DELGKT4_lungkuta(z,0.1,0,1,1,[x y])
yy =  1.0000
      1.1048
      1.2188
      1.3411
      1.4711
      1.6082
      1.7520
      1.9021
      2.0580
      2.2195
      2.3863
>> yy = DELGKT4_jer(z,0.1,0,1,1,[x y])
yy =  1.0000
      1.1048
      1.2188
      1.3411
      1.4711
      1.6082
      1.7520
      1.9021
      2.0580
      2.2195
      2.3863
>> yy = DELGKT4_qt(z,0.1,0,1,1,[x y])
yy =  1.0000
      1.1048
      1.2188
      1.3411
      1.4711
      1.6082
      1.7520
      1.9021
      2.0580
      2.2195
      2.3863
```

通过简单计算，可得出常微分方程

$$\begin{cases}\dfrac{dy}{dx}=1+\ln(x+1) \quad 0\leqslant x\leqslant 1\\ y(0)=1\end{cases}$$

的解为 $y=(1+x)\ln(1+x)+1$，将数值解与理论解列成表格如下：

x	经典法	基尔法	变型格式	理论解
0	1.0000	1.0000	1.0000	1.0000
0.1	1.1048	1.1048	1.1048	1.1048
0.2	1.2188	1.2188	1.2188	1.2188
0.3	1.3411	1.3411	1.3411	1.3411
0.4	1.4711	1.4711	1.4711	1.4711
0.5	1.6082	1.6082	1.6082	1.6082
0.6	1.7520	1.7520	1.7520	1.7520
0.7	1.9021	1.9021	1.9021	1.9021
0.8	2.0580	2.0580	2.0580	2.0580
0.9	2.2195	2.2195	2.2195	2.2195
1.0	2.3863	2.3863	2.3863	2.3863

可见四阶龙格-库塔法的精度已很高了，用它来解一般的常微分方程已经足够。

15.2.4 罗赛布诺克半隐式公式

罗赛布诺克半隐式法本质上是一种半隐式的龙格-库塔算法，它的迭代公式如下所示：

$$\begin{cases}y_{n+1}=y_n+w_1k_1+w_2k_2\\ k_1=h[1-ha_1f_y(x_n,y_n)]^{-1}f(x_n,y_n)\\ k_2=h[1-ha_2f_y(x_n+c_1h,y_n+c_2k_1)]^{-1}f(x_n+d_1h,y_n+d_2k_1)\end{cases}$$

其中，

$$a_1=1.40824829,\qquad a_2=0.59175171$$
$$d_1=c_1=0.17378667,\qquad d_2=c_2=0.17378667$$
$$w_1=-0.41315432,\qquad w_2=1.41315432$$

在 MATLAB 中编程实现的罗赛布诺克半隐式法的函数为：DELSBRK

功能：用罗赛布诺克半隐式法求一阶常微分方程的数值解

调用格式：y = DELSBRK (f, h,a,b,y0,varvec)

其中，f：一阶常微分方程的一般表达式的右端函数；

h：积分步长；

a：自变量取值下限；

b：自变量取值上限；

y0：函数初值；

varvec：常微分方程的变量组。

罗赛布诺克半隐式法的 MATLAB 程序代码如下所示：

```
function y = DELSBRK (f, h,a,b,y0,varvec)
%一阶常微分方程的一般表达式的右端函数：f
%积分步长：h
%自变量取值下限：a
%自变量取值上限：b
%函数初值：y0
%积分步长：h
%常微分方程的变量组：varvec
format long;
a1 = 1.40824829;
a2 = 0.59175171;
c1 = 0.17378667;
c2 = c1;
w1 = -0.41315432;
w2 = 1.41315432;
N = (b-a)/h;
y = zeros(N+1,1);
y(1) = y0;
x = a:h:b;
var = findsym(f);
dy = diff(f, varvec(2));
for i=2:N+1
   f1 = Funval(f,varvec,[x(i-1) y(i-1)]);
   dy1 = Funval(dy,varvec,[x(i-1) y(i-1)]);
    k1 = h*f1/(1-h*a1*dy1);    %第一个系数
    dy2 = Funval(dy,varvec,[x(i-1)+c1*h y(i-1)+c2*k1]);
    f2 = Funval(f,varvec,[x(i-1)+c1*h y(i-1)+c2*k1]);
    k2 = h*f2/(1-h*a2*dy2);    %第二个系数
    y(i) = y(i-1)+w1*k1+w2*k2;
end
format short;
```

例 15-7 罗赛布诺克法求解一阶常微分方程应用实例。用罗赛布诺克法求下面常微分方程的数值解。

$$\begin{cases} \dfrac{\mathrm{d}y}{\mathrm{d}x}=2xy \quad 0\leqslant x\leqslant 1 \\ y(0)=1 \end{cases}$$

解： 在 MATLAB 命令窗口中输入：

```
>> z = 2*x*y;
>> yy = DELSBRK(z, 0.1,0,1,1,[x y])
yy =  1.0000
      1.0049
      1.0303
      1.0777
      1.1501
      1.2522
```

```
1.3910
1.5765
1.8228
2.1501
2.5875
```

通过简单计算，可得出常微分方程

$$\begin{cases} \dfrac{\mathrm{d}y}{\mathrm{d}x} = 2xy \quad 0 \leqslant x \leqslant 1 \\ y(0) = 1 \end{cases}$$

的解为 $y = \mathrm{e}^{x^2}$，将数值解与理论解列成表格如下：

x	数值解	理论解
0	1.0000	1.0000
0.1	1.0049	1.0101
0.2	1.0303	1.0408
0.3	1.0777	1.0942
0.4	1.1501	1.1735
0.5	1.2522	1.2840
0.6	1.3910	1.4333
0.7	1.5765	1.6323
0.8	1.8228	1.8965
0.9	2.1501	2.2479
1.0	2.5875	2.7183

可见数值结果的误差还是比较大。

15.3 默森单步法

默森单步法的迭代公式如下所示：

$$\begin{cases} z_1 = y_n + \dfrac{h}{3} f(x_n, y_n) \\ z_2 = z_1 + \dfrac{h}{6}[f(x_n + \dfrac{h}{3}, z_1) - f(x_n, y_n)] \\ z_3 = z_2 + \dfrac{3h}{8}[f(x_n + \dfrac{h}{3}, z_2) - \dfrac{4}{9} f(x_n + \dfrac{h}{3}, z_1) - \dfrac{1}{9} f(x_n, y_n)] \\ z_4 = z_3 + 2h[f(x_n + \dfrac{h}{2}, z_3) - \dfrac{15}{16} f(x_n + \dfrac{h}{3}, z_2) + \dfrac{3}{16} f(x_n, y_n)] \\ y_{n+1} = z_4 + \dfrac{h}{6}[f(x_n + h, z_4) - 8f(x_n + \dfrac{h}{2}, z_3) + 9f(x_n + \dfrac{h}{3}, z_2) - 2f(x_n, y_n)] \end{cases}$$

在 MATLAB 中编程实现的默森单步法的函数为：DEMS

功能：用默森单步法求一阶常微分方程的数值解

调用格式：y = DEMS (f, h,a,b,y0,varvec)

其中，f：一阶常微分方程的一般表达式的右端函数；

h：积分步长；

a：自变量取值下限；

b：自变量取值上限；

y0：函数初值；

varvec：常微分方程的变量组。

默森单步法的 MATLAB 程序代码如下所示：

```
 function y = DEMS (f, h,a,b,y0,varvec)
%一阶常微分方程的一般表达式的右端函数：f
%积分步长：h
%自变量取值下限：a
%自变量取值上限：b
%函数初值：y0
%积分步长：h
%常微分方程的变量组：varvec
function y = DEMS(f, h,a,b,y0,varvec)
format long;
N = (b-a)/h;
y = zeros(N+1,1);
y(1) = y0;
x = a:h:b;
var = findsym(f);
for i=2:N+1
    fy = Funval(f,varvec,[x(i-1) y(i-1)]);
    z1 = y(i-1)+h*fy/3;   %第一步迭代值
    fy1 = Funval(f,varvec,[x(i-1)+h/3 z1]);
    z2 = z1+h*(fy1-fy)/6; %第二步迭代值
    fy2 = Funval(f,varvec,[x(i-1)+h/3 z2]);
    z3 = z2+3*h*(fy2 - 4*fy1/9-fy/9)/8; %第三步迭代值
    fy3 = Funval(f,varvec ,[x(i-1)+h/2 z3]);
    z4 = z3+2*h*(fy3- 15*fy2/16 + 3*fy/16); %第四步迭代值
    fy4 = Funval(f,varvec ,[x(i-1)+h z4]);
    y(i) = z4+h*(fy4 - 8*fy3 + 9*fy2 - 2*fy)/6; %第五步迭代值
end
format short;
```

例 15-8 默森单步法求解一阶常微分方程应用实例。用默森单步法求下面常微分方程的数值解。

$$\begin{cases}\dfrac{\mathrm{d}y}{\mathrm{d}x}=2xy, 0\leqslant x\leqslant 1\\ y(0)=1\end{cases}$$

解： 在 MATLAB 命令窗口中输入：

```
>> z = 2*x*y;
>> yy = DEMS(z, 0.1,0,1,1,[x y])
yy =  1.0000
      1.0101
      1.0408
      1.0942
      1.1735
      1.2840
      1.4333
      1.6323
      1.8965
      2.2479
      2.7183
```

与上一例的理论解比较，可以看出默森单步法的结果比罗赛布诺克法的结果要好得多。

15.4 线性多步法

常用的线性多步法有米尔恩法和亚当斯法。

- 米尔恩法的计算公式为：

$$y_{n+4} = y_n + \frac{4h}{3}(2f_{n+1} - f_{n+2} + 2f_{n+3})$$

在 MATLAB 中编程实现的米尔恩法的函数为：DEMiren

功能：用米尔恩法求一阶常微分方程的数值解

调用格式：y = DEMiren (f, h,a,b,y0,varvec)

其中，f：一阶常微分方程的一般表达式的右端函数；

h：积分步长；

a：自变量取值下限；

b：自变量取值上限；

y0：函数初值；

varvec：常微分方程的变量组。

米尔恩法的 MATLAB 程序代码如下所示：

```
 function y = DEMiren (f, h,a,b,y0,varvec)
%一阶常微分方程的一般表达式的右端函数：f
%积分步长：h
%自变量取值下限：a
%自变量取值上限：b
%函数初值：y0
%常微分方程的变量组：varvec
function y = DEMiren(f, h,a,b,y0,varvec)
format long;
```

```
N = (b-a)/h;
y = zeros(N+1,1);
x = a:h:b;
y(1) = y0;
y(2) = y0 + h*Funval(f,varvec, [x(1) y(1)]);
y(3) = y(2) + h*Funval(f,varvec, [x(2) y(2)]);
y(4) = y(3) + h*Funval(f,varvec, [x(3) y(3)]);
var = findsym(f);
for i=5:N+1          %米尔恩法的递推公式
   y(i) = y(i-4)+4*h*(2*Funval(f,varvec,[x(i-1), y(i-1)]) - …
                  Funval(f,varvec,[x(i-2), y(i-2)]) + …
                  2*Funval(f,varvec,[x(i-3), y(i-3)]))/3;
end
format short;
```

例 15-9 米尔恩法求解一阶常微分方程应用实例。用米尔恩法求下面常微分方程的数值解。

$$\begin{cases} \dfrac{dy}{dx} = 2xy \quad 0 \leqslant x \leqslant 1 \\ y(0) = 1 \end{cases}$$

解： 在 MATLAB 命令窗口中输入：

```
>> z = 2*x*y;
>> yy = DEMiren(z, 0.1,0,1,1,[x y])
yy =  1.0000
      1.0000
      1.0200
      1.0608
      1.1687
      1.2733
      1.4046
      1.5898
      1.8770
      2.2268
      2.6666
```

- 亚当斯法的计算公式为：

亚当斯法是一种插值型的多步法，它的一般递推公式为：

$$y_{n+k} = y_{n+k-1} + h\sum_{r=1}^{p} c_r f_{n+k-r}$$

其中的系数如下表所示：

p	c_1	c_2	c_3	c_4	c_5
1	1				
2	$\frac{3}{2}$	$-\frac{1}{2}$			
3	$\frac{23}{12}$	$-\frac{4}{3}$	$\frac{5}{12}$		

续表

p	c_1	c_2	c_3	c_4	c_5
4	$\frac{55}{24}$	$-\frac{59}{24}$	$\frac{37}{24}$	$-\frac{9}{24}$	
5	$\frac{1901}{720}$	$-\frac{2774}{720}$	$\frac{2616}{720}$	$-\frac{1274}{720}$	$\frac{251}{720}$

在 MATLAB 中编程实现的亚当斯法的函数为：DEYDS

功能：用亚当斯法求一阶常微分方程的数值解

调用格式：y = DEYDS (f, h,a,b,y0,varvec)

其中，f：一阶常微分方程的一般表达式的右端函数；

h：积分步长；

a：自变量取值下限；

b：自变量取值上限；

y0：函数初值；

varvec：常微分方程的变量组。

亚当斯法的 MATLAB 程序代码如下所示：

```
function y = DEYDS (f, h,a,b,y0,varvec)
%一阶常微分方程的一般表达式的右端函数：f
%积分步长：h
%自变量取值下限：a
%自变量取值上限：b
%函数初值：y0
%常微分方程的变量组：varvec

format long;
N = (b-a)/h;
y = zeros(N+1,1);
x = a:h:b;
y(1) = y0;
y(2) = y0 + h*Funval(f,varvec, [x(1) y(1)]);
y(3) = y(2) + h*Funval(f,varvec, [x(2) y(2)]);
y(4) = y(3) + h*Funval(f,varvec, [x(3) y(3)]);
y(5) = y(4) + h*Funval(f,varvec, [x(4) y(4)]);
var = findsym(f);
for i=6:N+1        %亚当斯法的递推公式
   y(i) = y(i-1)+h*(55*Funval(f,varvec,[x(i-1), y(i-1)])/24 - …
                    59*Funval(f,varvec,[x(i-2), y(i-2)])/24 + …
                    37*Funval(f,varvec,[x(i-3), y(i-3)])/24 - …
                    9*Funval(f,varvec,[x(i-4), y(i-4)])/24);
end
format short;
```

例 15-10　亚当斯法求解一阶常微分方程应用实例。用亚当斯法求下面常微分方程的数值解。

$$\begin{cases}\dfrac{dy}{dx}=2xy & 0\leqslant x\leqslant 1\\ y(0)=1\end{cases}$$

解： 在 MATLAB 命令窗口中输入：

```
>> z = 2*x*y;
>> yy = DEYDS(z, 0.1,0,1,1,[x y])
yy =  1.0000
      1.0000
      1.0200
      1.0608
      1.1244
      1.2295
      1.3730
      1.5631
      1.8154
      2.1510
      2.5998
```

15.5 预测-校正法

预测-校正法是一种显式法和隐式法相结合的方法，其算法过程介绍如下。

❶ 首先用任一个显式算法计算出下一个 y_{n+k} 的估计值 $\tilde{y}$（预测）；

❷ 再将 $\tilde{y}$ 代替 y_{n+k} 代入一个隐式算法的右边，计算得到 y_{n+k} 的校正值。

因此预测-校正法可以有很多种形式，常见的有中点-梯形、阿达姆斯、密伦、亚当斯和汉明预测-校正法，下面分别进行讲述。

15.5.1 中点-梯形预测-校正法

把中点公式作为预测公式，梯形公式作为校正公式得到中点-梯形公式，它有三种模式。令 $f_n=f(x_n,y_n)$，$\tilde{f}_n=f(x_n,\tilde{y}_n)$

- PC 模式的计算公式为：

$$\begin{cases}\tilde{y}_{n+1}=y_{n-1}+2hf(x_n,y_n)\\ y_{n+1}=y_n+\dfrac{h}{2}[f_n+\tilde{f}_{n+1}]\end{cases}$$

- 迭代模式的计算公式为：

$$\begin{cases}y_{n+1}^{(0)}=y_{n-1}+2hf_n\\ f_{n+1}^{(0)}=f(x_{n+1},y_{n+1}^{(0)})\\ y_{n+1}^{(s)}=y_n+\dfrac{h}{2}(f_{n+1}^{(s-1)}+f_n)\\ f_{n+1}^{(s)}=f(x_{n+1},y_{n+1}^{(s)})\end{cases}$$

其中 $s=1,2,\cdots$

- 修改模式的计算公式为：

$$\begin{cases} p_{n+1} = y_{n-1} + 2hf_n \\ M_{n+1} = p_{n+1} - 0.8(p_n - c_n) \\ F_{n+1} = f(x_{n+1}, M_{n+1}) \\ f_{n+1} = y_n + \dfrac{h}{2}(f_n + F_{n+1}) \\ y_{n+1} = f_{n+1} - 0.2(p_{n+1} - f_{n+1}) \\ c_{n+1} = f_{n+1} \end{cases}$$

在开始计算时，取 $p_0 = c_0 = 0$。

在 MATLAB 中编程实现的中点-梯形预测-校正法的函数为：DEYCJZ_mid

功能：用中点-梯形预测-校正法求一阶常微分方程的数值解

调用格式：y = DEYCJZ_mid (f, h,a,b,y0,varvec,type,s)

其中，f：一阶常微分方程的一般表达式的右端函数；

h：积分步长；

a：自变量取值下限；

b：自变量取值上限；

y0：函数初值；

varvec：常微分方程的变量组；

type：中点-梯形预测-校正法的类型；

s：第二类中点-梯形预测-校正法的迭代步数。

中点-梯形预测-校正法的 MATLAB 程序代码如下所示：

```
function y = DEYCJZ_mid(f, h,a,b,y0,varvec,type,s)
%一阶常微分方程的一般表达式的右端函数：f
%积分步长：h
%自变量取值下限：a
%自变量取值上限：b
%函数初值：y0
%常微分方程的变量组：varvec
%中点-梯形预测-校正法的类型：type
%第二类中点-梯形预测-校正法的迭代步数：s
format long;
N = (b-a)/h;
y = zeros(N+1,1);
x = a:h:b;
y(1) = y0;
y(2) = y0+h*Funval(f,varvec,[x(1) y(1)]);
var = findsym(f);
if type == 1     %PC 模式
    for i=3:N+1
        v1 = Funval(f,varvec,[x(i-1) y(i-1)]);
        t = y(i-2) + 2*h*v1;
```

```
            v2 = Funval(f,varvec,[x(i) t]);
            y(i) = y(i-1)+h*(v1+v2)/2;
        end
    else
        if type == 2    %迭代模式
            for i=3:N+1
                v1 = Funval(f,varvec,[x(i-1) y(i-1)]);
                t = y(i-2) + 2*h*v1;
                v2 = Funval(f,varvec,[x(i) t]);
                for l=1:s
                    y(i) = y(i-1)+h*(v2 + v1)/2;
                    v2 = Funval(f,varvec,[x(i) y(i)]);
                end
            end
        else            %修改模式
            p0 = 0;
            c = 0;
            fnl = Funval(f,varvec,[x(1) y(1)]);
            for i = 3:N+1
                v1 = Funval(f,varvec,[x(i-1) y(i-1)]);
                p = y(i-2)+2*h*v1;
                M = p - 0.8*(p0 - c);
                F = Funval(f , varvec, [x(i) ,M]);
                fn = y(i-1) + h*(fnl + F)/2;
                y(i) = fn - 0.2*( p - fn);
                c = fn;
                fnl = fn;
                p0 = p;
            end
        end
    end
    format short;
```

例 15-11 中点-梯形预测-校正法求解一阶常微分方程应用实例。用中点-梯形预测-校正法求下面常微分方程的数值解。

$$\begin{cases} \dfrac{dy}{dx} = 2xy \quad 0 \leqslant x \leqslant 1 \\ y(0) = 1 \end{cases}$$

解： 在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = 2*x*y;
>> y= DEYCJZ_mid (z, 0.1,0,1,1,[x y],1,3)
y =  1.0000
     1.0000
     1.0308
     1.0839
     1.1628
     1.2729
```

```
     1.4215
     1.6198
     1.8832
     2.2339
     2.7037
>> y= DEYCJZ_mid (z, 0.1,0,1,1,[x y],2,3)
y =  1.0000
     1.0000
     1.0306
     1.0837
     1.1628
     1.2729
     1.4219
     1.6206
     1.8849
     2.2370
     2.7092
>> y= DEYCJZ_mid (z, 0.1,0,1,1,[x y],3,4)
y =  1.0000
     1.0000
     1.0170
     1.1037
     1.2164
     1.3530
     1.5165
     1.7130
     1.9505
     2.2398
     2.5948
```

第一种类型的中点-梯形预测-校正法和第二类型得出的结果比较接近，且精度相对要高一些。

15.5.2　阿达姆斯预测-校正法

阿达姆斯预测-校正法的公式如下所示：

$$\begin{cases} \tilde{y}_{n+1} = y_n + \dfrac{h}{2}(3f_n - f_{n-1}) \\ y_{n+1} = y_n + \dfrac{h}{2}[f_n + \tilde{f}_{n+1}] \end{cases}$$

在 MATLAB 中编程实现的阿达姆斯预测-校正法的函数为：DEYCJZ_adms

功能：用阿达姆斯预测-校正法求一阶常微分方程的数值解

调用格式：y = DEYCJZ_adms (f, h,a,b,y0,varvec)

其中，f：一阶常微分方程的一般表达式的右端函数；

　　h：积分步长；

　　a：自变量取值下限；

b：自变量取值上限；
y0：函数初值；
varvec：常微分方程的变量组。

阿达姆斯预测-校正法的 MATLAB 程序代码如下所示：

```
function y = DEYCJZ_adms (f, h,a,b,y0,varvec)
%一阶常微分方程的一般表达式的右端函数：f
%积分步长：h
%自变量取值下限：a
%自变量取值上限：b
%函数初值：y0
%常微分方程的变量组：varvec
format long;
N = (b-a)/h;
y = zeros(N+1,1);
x = a:h:b;
y(1) = y0;
y(2) = y0+h*Funval(f,varvec,[x(1) y(1)]);
for i=3:N+1
      v1 = Funval(f,varvec,[x(i-2) y(i-2)]);
      v2 = Funval(f,varvec,[x(i-1) y(i-1)]);
      t = y(i-1) + h*(3*v2-v1)/2;
      v3 = Funval(f,varvec,[x(i) t]);
      y(i) = y(i-1)+h*(v2+v3)/2;   %阿达姆斯预测-校正法的递推公式
end
format short;
```

例 15-12 阿达姆斯预测-校正法求解一阶常微分方程应用实例。用阿达姆斯预测-校正法求下面常微分方程的数值解。

$$\begin{cases} \dfrac{dy}{dx} = x - y + 1 & 0 \leqslant x \leqslant 1 \\ y(0) = 1 \end{cases}$$

解：在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = x-y+1;
>> y = DEYCJZ_adms(z,0.1,0,1,1,[x y])
y =  1.0000
     1.0000
     1.0143
     1.0367
     1.0665
     1.1030
     1.1456
     1.1936
     1.2466
     1.3040
     1.3655
```

将数值解与理论解比较如下：

x	阿达姆斯预测-校正法	理论解	误 差
0	1.0000	1.0000	0
0.1	1.0000	1.0048	0.0048
0.2	1.0143	1.0187	0.0044
0.3	1.0367	1.0408	0.0041
0.4	1.0665	1.0703	0.0038
0.5	1.1030	1.1065	0.0035
0.6	1.1456	1.1488	0.0032
0.7	1.1936	1.1966	0.0030
0.8	1.2466	1.2493	0.0027
0.9	1.3040	1.3066	0.0026
1.0	1.3655	1.3679	0.0024

从误差列可以看出，阿达姆斯预测-校正法使得误差越来越小。

15.5.3 密伦预测-校正法

密伦预测-校正法的公式如下所示：

$$\begin{cases} y_{n+1}^{(0)} = y_{n-3} + \dfrac{4h}{3}(2f_n - f_{n-1} + 2f_{n-2}) \\ f_{n+1}^{(0)} = f(x_{n+1}, y_{n+1}^{(0)}) \\ y_{n+1}^{(s)} = y_{n-1} + \dfrac{h}{3}(f_{n+1}^{(s-1)} + 4f_n + f_{n-1}) \\ f_{n+1}^{(s)} = f(x_{n+1}, y_{n+1}^{(s)}) \end{cases}$$

其中 $s = 1, 2, \cdots$

还有修正的密伦预测-校正法，其公式为：

$$\begin{cases} p_{n+1} = y_{n-3} + \dfrac{4h}{3}(2f_n - f_{n-1} + 2f_{n-2}) \\ M_{n+1} = p_{n+1} - \dfrac{28}{29}(p_n - c_n) \\ F_{n+1} = f(x_{n+1}, M_{n+1}) \\ c_{n+1} = y_{n-2} + \dfrac{h}{3}(F_{n+1} + 4f_n + f_{n-1}) \\ y_{n+1} = c_{n+1} + \dfrac{1}{29}(p_{n+1} - c_{n+1}) \end{cases}$$

在 MATLAB 中编程实现的密伦预测-校正法的函数为：DEYCJZ_adms2

功能：用密伦预测-校正法求一阶常微分方程的数值解

调用格式：y = DEYCJZ_adms2 (f, h,a,b,y0,varvec,type,s)

其中，f：一阶常微分方程的一般表达式的右端函数；
　　h：积分步长；
　　a：自变量取值下限；
　　b：自变量取值上限；
　　y0：函数初值；
　　varvec：常微分方程的变量组；
　　type：密伦预测-校正法的类型；
　　s：迭代参数。

密伦预测-校正法的 MATLAB 程序代码如下所示：

```
function y = DEYCJZ_adms2 (f, h,a,b,y0,varvec,type,s)
%一阶常微分方程的一般表达式的右端函数：f
%积分步长：h
%自变量取值下限：a
%自变量取值上限：b
%函数初值：y0
%常微分方程的变量组：varvec
%密伦预测-校正法的类型：type
%迭代参数：s
function y = DEYCJZ_ml(f, h,a,b,y0,varvec,type)
format long;
N = (b-a)/h;
y = zeros(N+1,1);
x = a:h:b;
y(1) = y0;
y(2) = y0+h*Funval(f,varvec,[x(1) y(1)]);
y(3) = y(2)+h*Funval(f,varvec,[x(2) y(2)]);
y(4) = y(3)+h*Funval(f,varvec,[x(3) y(3)]);
if type == 1      %密伦预测-校正公式
    for i=5:N+1
        v1 = Funval(f,varvec,[x(i-3) y(i-3)]);
        v2 = Funval(f,varvec,[x(i-2) y(i-2)]);
        v3 = Funval(f,varvec,[x(i-1) y(i-1)]);
        t = y(i-4) + 4*h*(2*v3 - v2 + 2*v1)/3;
        for m=1:s
            ft = Funval(f,varvec,[x(i) t]);
            y(i) = y(i-2)+h*(4*v3 + v2 + ft)/3;
            t = y(i);
        end
    end
else              %修正的密伦预测-校正公式
    p0 = 0;
    c = 0;
    fnl = Funval(f,varvec,[x(1) y(1)]);
    v1 = Funval(f,varvec,[x(i-3) y(i-3)]);
    v2 = Funval(f,varvec,[x(i-2) y(i-2)]);
```

```
    v3 = Funval(f,varvec,[x(i-1) y(i-1)]);
    t = y(i-4) + 4*h*(2*v3 - v2 + 2*v1)/3;
    ft = Funval(f,varvec,[x(i) t]);
    for i = 5:N+1
       p = y(i-4)+4*h*(2*v3 - v2 + 2*v1)/3;
       M = p - 28*(p0 - c)/29;
       F = Funval(f , varvec, [x(i) ,M]);
       c = y(i-2)+h*(4*v3 + v2 + F)/3;
       y(i) = c + ( p - c)/29;
       p0 = p;
    end
end
format short;
```

例 15-13 密伦预测-校正法求解一阶常微分方程应用实例。用密伦预测-校正法求下面常微分方程的数值解。

$$\begin{cases}\dfrac{dy}{dx}=x-y+1 \quad 0\leqslant x\leqslant 1\\ y(0)=1\end{cases}$$

解： 在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = x-y+1;
>> yy = DEYCJZ_ml(z,0.1,0,1,1,[x y],1,4)
yy =  1.0000
      1.0000
      1.0100
      1.0290
      1.0637
      1.0963
      1.1439
      1.1877
      1.2459
      1.2987
      1.3657
>> yy = DEYCJZ_ml(z,0.1,0,1,1,[x y],2)
yy =  1.0000
      1.0000
      1.0100
      1.0290
      1.0637
      1.0967
      1.1441
      1.1880
      1.2460
      1.2990
      1.3657
```

从结果看，密伦预测-校正法的两种形式算出的结果差别非常小。

15.5.4 亚当斯预测-校正法

亚当斯预测-校正法的公式如下所示：

$$\begin{cases} \tilde{y}_{n+1} = y_n + \dfrac{h}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}) \\ y_{n+1} = y_n + \dfrac{h}{24}[9\tilde{f}_{n+1} + 19f_n - 5f_{n-1} + f_{n-2})] \end{cases}$$

在 MATLAB 中编程实现的亚当斯预测-校正法的函数为：DEYCJZ_ yds

功能：用亚当斯预测-校正法求一阶常微分方程的数值解

调用格式：y = DEYCJZ_yds (f, h,a,b,y0,varvec)

其中，f：一阶常微分方程的一般表达式的右端函数；

h：积分步长；

a：自变量取值下限；

b：自变量取值上限；

y0：函数初值；

varvec：常微分方程的变量组。

亚当斯预测-校正法的 MATLAB 程序代码如下所示：

```
function y = DEYCJZ_ yds (f, h,a,b,y0,varvec)
%一阶常微分方程的一般表达式的右端函数：f
%积分步长：h
%自变量取值下限：a
%自变量取值上限：b
%函数初值：y0
%常微分方程的变量组：varvec
format long;
N = (b-a)/h;
y = zeros(N+1,1);
x = a:h:b;
y(1) = y0;
y(2) = y0+h*Funval(f,varvec,[x(1) y(1)]);
y(3) = y(2)+h*Funval(f,varvec,[x(2) y(2)]);
y(4) = y(3)+h*Funval(f,varvec,[x(3) y(3)]);
for i=5:N+1
    v1 = Funval(f,varvec,[x(i-4) y(i-4)]);
    v2 = Funval(f,varvec,[x(i-3) y(i-3)]);
    v3 = Funval(f,varvec,[x(i-2) y(i-2)]);
    v4 = Funval(f,varvec,[x(i-1) y(i-1)]);
    t = y(i-1) + h*(55*v4 - 59*v3 + 37*v2 - 9*v1)/24;
    ft = Funval(f,varvec,[x(i) t]);
    y(i) = y(i-1)+h*(9*ft+19*v4-5*v3+v2)/24; %亚当斯预测-校正法的递推公式
end
```

```
format short;
```

例 15-14 亚当斯预测-校正法求解一阶常微分方程应用实例。用亚当斯预测-校正法求下面常微分方程的数值解。

$$\begin{cases} \dfrac{dy}{dx} = 1 + \cos x & 0 \leqslant x \leqslant 1 \\ y(0) = 0 \end{cases}$$

解：在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = 1+cos(x);
>> y = DEYCJZ_yds(z,0.1,0,1,0,[x y])
y =      0
    0.2000
    0.3995
    0.5975
    0.7914
    0.9814
    1.1666
    1.3462
    1.5193
    1.6853
    1.8435
```

可以看出，常微分方程

$$\begin{cases} \dfrac{dy}{dx} = 1 + \cos x & 0 \leqslant x \leqslant 1 \\ y(0) = 0 \end{cases}$$

的理论解为 $y = x + \sin x$，数值解和理论解的结果比较如下：

x	亚当斯预测-校正法	理论解
0	0.0000	0.0000
0.1	0.2000	0.1998
0.2	0.3995	0.3987
0.3	0.5975	0.5955
0.4	0.7914	0.7894
0.5	0.9814	0.9794
0.6	1.1666	1.1646
0.7	1.3462	1.3442
0.8	1.5193	1.5174
0.9	1.6853	1.6833
1.0	1.8435	1.8415

亚当斯预测-校正法的计算结果精度还是可以的。

另外还有修正的亚当斯预测-校正法，其公式如下所示：

$$\begin{cases} p_{n+1} = y_n + \dfrac{h}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}) \\ M_{n+1} = p_{n+1} + \dfrac{251}{270}(c_n - p_n) \\ F_{n+1} = f(x_{n+1}, M_{n+1}) \\ c_{n+1} = y_n + \dfrac{h}{24}[9F_{n+1} + 19f_n - (5f_{n-1} + f_{n-2})] \\ y_{n+1} = c_{n+1} + \dfrac{19}{270}(p_{n+1} - c_{n+1}) \end{cases}$$

在 MATLAB 中编程实现的修正的亚当斯预测-校正法的函数为：DEYCJZ_ myds

功能：用修正的亚当斯预测-校正法求一阶常微分方程的数值解

调用格式：y = DEYCJZ_myds (f, h,a,b,y0,varvec)

其中，f：一阶常微分方程的一般表达式的右端函数；

h：积分步长；

a：自变量取值下限；

b：自变量取值上限；

y0：函数初值；

varvec：常微分方程的变量组。

修正的亚当斯预测-校正法的 MATLAB 程序代码如下所示：

```
function y = DEYCJZ_myds (f, h,a,b,y0,varvec)
%一阶常微分方程的一般表达式的右端函数：f
%积分步长：h
%自变量取值下限：a
%自变量取值上限：b
%函数初值：y0
%常微分方程的变量组：varvec
function y = DEYCJZ_myds(f, h,a,b,y0,varvec,type)
format long;
N = (b-a)/h;
y = zeros(N+1,1);
x = a:h:b;
y(1) = y0;
y(2) = y0+h*Funval(f,varvec,[x(1) y(1)]);
y(3) = y(2)+h*Funval(f,varvec,[x(2) y(2)]);
y(4) = y(3)+h*Funval(f,varvec,[x(3) y(3)]);
p0 = 0;
c = 0;
for i=5:N+1
    v1 = Funval(f,varvec,[x(i-4) y(i-4)]);
    v2 = Funval(f,varvec,[x(i-3) y(i-3)]);
    v3 = Funval(f,varvec,[x(i-2) y(i-2)]);
```

```
    v4 = Funval(f,varvec,[x(i-1) y(i-1)]);
    p = y(i-1) + h*(55*v4 - 59*v3 + 37*v2 - 9*v1)/24;
    M = p + 251*(c - p0)/270;
    F = Funval(f,varvec,[x(i) M]);
    c = y(i-1) + h*(9*F + 19*v4 - 5*v3 + v2)/24;
    y(i) = c+h*19*(p - c)/270;   %修正的亚当斯预测-校正法的递推公式
    p0 = p;
end
format short;
```

例 15-15　修正的亚当斯预测-校正法求解一阶常微分方程应用实例。用修正的亚当斯预测-校正法求下面常微分方程的数值解。

$$\begin{cases} \dfrac{\mathrm{d}y}{\mathrm{d}x} = 1+\cos x & 0 \leqslant x \leqslant 1 \\ y(0) = 0 \end{cases}$$

解： 在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = 1+cos(x);
>> y = DEYCJZ_myds(z,0.1,0,1,0,[x y])
y =      0
    0.2000
    0.3995
    0.5975
    0.7914
    0.9814
    1.1666
    1.3462
    1.5193
    1.6853
    1.8435
```

此题的理论解为 $y = x + \sin x (0 \leqslant x \leqslant 1)$，数值解与理论解的比较结果如下：

x	理论解	修正的亚当斯预测-校正法
0	0	0
0.1	0.1998	0.2000
0.2	0.3987	0.3995
0.3	0.5955	0.5975
0.4	0.7894	0.7914
0.5	0.9794	0.9814
0.6	1.1646	1.1666
0.7	1.3442	1.3462
0.8	1.5174	1.5193
0.9	1.6833	1.6853
1	1.8415	1.8435

从上表可以看出，修正的亚当斯预测-校正法的计算结果精度还是可以的。

15.5.5 汉明预测-校正法

汉明预测-校正法的公式如下所示：

$$\begin{cases} \tilde{y}_{n+1} = y_{n-3} + \dfrac{4h}{3}(2f_n - f_{n-1} + 2f_{n-2}) \\ y_{n+1} = \dfrac{1}{8}[9y_n - y_{n-2} + 3h(\tilde{f}_{n+1} + 2f_n - f_{n-1})] \end{cases}$$

修正的汉明法公式如下所示：

$$\begin{cases} p_{n+1} = y_{n-3} + \dfrac{4h}{3}(2f_n - f_{n-1} + 2f_{n-2}) \\ M_{n+1} = p_{n+1} - \dfrac{112}{121}(p_n - c_n) \\ F_{n+1} = f(x_{n+1}, M_{n+1}) \\ c_{n+1} = \dfrac{1}{8}[9y_n - y_{n-2} + 3h(F_{n+1} + 2f_n - f_{n-1})] \\ y_{n+1} = c_{n+1} + \dfrac{9}{121}(p_{n+1} - c_{n+1}) \end{cases}$$

在 MATLAB 中编程实现的汉明预测-校正法的函数为：DEYCJZ_hm

功能：用汉明预测-校正法求一阶常微分方程的数值解

调用格式：y = DEYCJZ_hm (f, h,a,b,y0,varvec,type)

其中，f：一阶常微分方程的一般表达式的右端函数；

h：积分步长；

a：自变量取值下限；

b：自变量取值上限；

y0：函数初值；

varvec：常微分方程的变量组；

type：汉明预测-校正法的类型。

汉明预测-校正法的 MATLAB 程序代码如下所示：

```
function y = DEYCJZ_hm (f, h,a,b,y0,varvec,type)
%一阶常微分方程的一般表达式的右端函数：f
%积分步长：h
%自变量取值下限：a
%自变量取值上限：b
%函数初值：y0
%常微分方程的变量组：varvec
%汉明预测-校正法的类型：type
function y = DEYCJZ_hm(f, h,a,b,y0,varvec,type)
```

```
format long;
N = (b-a)/h;
y = zeros(N+1,1);
x = a:h:b;
y(1) = y0;
y(2) = y0+h*Funval(f,varvec,[x(1) y(1)]);
y(3) = y(2)+h*Funval(f,varvec,[x(2) y(2)]);
y(4) = y(3)+h*Funval(f,varvec,[x(3) y(3)]);
if type == 1        %汉明预测-校正法
   for i=5:N+1
      v1 = Funval(f,varvec,[x(i-3) y(i-3)]);
      v2 = Funval(f,varvec,[x(i-2) y(i-2)]);
      v3 = Funval(f,varvec,[x(i-1) y(i-1)]);
      t = y(i-4) + 4*h*(2*v3 - v2 + 2*v1)/3;
      ft = Funval(f,varvec,[x(i) t]);
      y(i) = (9*y(i-1) -y(i-3) +3*h*(2*v3 + ft-v2))/8;
   end
else                %修正的汉明预测-校正法
   p0 = 0;
   c = 0;
   fn1 = Funval(f,varvec,[x(1) y(1)]);
   v1 = Funval(f,varvec,[x(i-3) y(i-3)]);
   v2 = Funval(f,varvec,[x(i-2) y(i-2)]);
   v3 = Funval(f,varvec,[x(i-1) y(i-1)]);
   t = y(i-4) + 4*h*(2*v3 - v2 + 2*v1)/3;
   ft = Funval(f,varvec,[x(i) t]);
   for i = 5:N+1
      p = y(i-4)+4*h*(2*v3 - v2 + 2*v1)/3;
      M = p - 112*(p0 - c)/121;
      F = Funval(f , varvec, [x(i) ,M]);
      c = (9*y(i-1) -y(i-3) +3*h*(2*v3 + F-v2))/8;
      y(i) = c + 9*( p - c)/121;
      p0 = p;
   end
end
format short;
```

例 15-16 汉明预测-校正法求解一阶常微分方程应用实例。用汉明预测-校正法求下面常微分方程的数值解。

$$\begin{cases} \dfrac{dy}{dx} = 1 + \cos x \quad 0 \leqslant x \leqslant 1 \\ y(0) = 0 \end{cases}$$

解： 在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = 1+cos(x);
>> yy = DEYCJZ_hm(z,0.1,0,1,0,[x y],1)
yy =    0
   0.2000
```

```
    0.3995
    0.5975
    0.7916
    0.9818
    1.1671
    1.3467
    1.5198
    1.6858
    1.8440
>> yy = DEYCJZ_hm(z,0.1,0,1,0,[x y],2)
yy =     0
    0.2000
    0.3995
    0.5975
    0.7915
    0.9815
    1.1666
    1.3462
    1.5193
    1.6853
    1.8434
```

此题的理论解为 $y=x+\sin x(0\leqslant x\leqslant 1)$ ，数值解与理论解的比较结果如下：

x	理论解	汉明预测-校正法 1	汉明预测-校正法 2
0	0	0	0
0.1	0.1998	0.2000	0.2000
0.2	0.3987	0.3995	0.3995
0.3	0.5955	0.5975	0.5975
0.4	0.7894	0.7916	0.7915
0.5	0.9794	0.9818	0.9815
0.6	1.1646	1.1671	1.1666
0.7	1.3442	1.3467	1.3462
0.8	1.5174	1.5198	1.5193
0.9	1.6833	1.6858	1.6853
1	1.8415	1.8440	1.8434

在此题中，汉明预测-校正法 2 的计算结果比汉明预测-校正法 1 的计算结果稍好一点。

15.6 外推法

外推技术是通过采用不同的步长求得微分方程的一系列数值解，然后通过外推公式来逼近微分方程的解。通用外推法给出了用外推法求解一般微分方程的求解步骤，下面给出了欧拉法的外推程序，格拉格外推法是外推法的一种特殊形式。

15.6.1　通用外推法

设初值问题

$$\begin{cases}\dfrac{\mathrm{d}y}{\mathrm{d}x}=f(x,y) & x_0\leqslant x\leqslant b\\ y(x_0)=y_0\end{cases}$$

的准确解为 $y(x)$，假设用某个步长 h_i 的数值方法计算 N_i 步到 $x=x_0+H$。其中 $H=N_ih_i$，并有：

$$y(x_0+H;h_i)=p_i$$

让 h_i 取 $h_0,h_1,h_2,\cdots$ 一系列值，得到

$$\begin{cases}y(x_0+H;h_0)=p_0\\ y(x_0+H;h_1)=p_1\\ y(x_0+H;h_2)=p_2\\ \qquad\vdots\end{cases}$$

外推法就是以上面的计算结果为基础，构造下面的计算表格：

$p_0^{(0)}$					
$p_1^{(0)}$	$p_0^{(1)}$				
$p_2^{(0)}$	$p_1^{(1)}$	$p_0^{(2)}$			
$p_3^{(0)}$	$p_2^{(1)}$	$p_1^{(2)}$	$p_0^{(3)}$		
⋮	⋮	⋮	⋮	⋮	⋮

其中 $y(x_0+H;h_i)=p_i^{(0)}$，$p_i^{(j)}=p_{i+1}^{(j-1)}+\dfrac{p_{i+1}^{(j-1)}-p_i^{(j-1)}}{(h_i/h_{i+j})^{\gamma}-1}\qquad(h_0>h_1>h_2>\cdots)$

则 $p_0^{(k)}$ 及表格对角线上的值可作为解的最好近似。

在 MATLAB 中编程实现的通用外推法的函数为：yy = DEWT(f,h,a,b,gama,y0,order,varvec)

功能：用外推法求一阶常微分方程的数值解

调用格式：yy = DEWT(f,h,a,b,gama,y0,order,varvec)

其中，f：一阶常微分方程的一般表达式的右端函数；

h：积分步长；

a：自变量取值下限；

b：自变量取值上限；

gama：外推参数，参考外推公式；

y0：函数初值；

order：外推阶数；

varvec：常微分方程的变量组。

外推法的 MATLAB 程序代码如下所示：

```
function yy = DEWT(f,h,a,b,gama,y0,order,varvec)
%一阶常微分方程的一般表达式的右端函数：f
%积分步长：h
%自变量取值下限：a
%自变量取值上限：b
%外推参数，参考外推公式：gama
%函数初值：y0
%外推阶数：order
%常微分方程的变量组：varvec
format long;
ArrayH = [1;2;4;6;8;12;16;24;32;48;64;96];
N = (b-a)/h;
yy = zeros(N+1,1);
for i = 2:N+1
    dh = h;
    s = zeros(order,1);
    for j=1:order
        dh = h/ArrayH(j);       %不同的 h 值
        tmpY = DELGKT2_suen(f,dh,a,a+(i-1)*h,y0,varvec);  %休恩法
        s(j) = tmpY((i-1)*ArrayH(j)+1);
    end
    tmpS = zeros(order,1);
    for j=1:order-1
        for k=(j+1):order
            tmpS(k) = s(k)+(s(k)-s(k-1))/((ArrayH(k)/ArrayH(j))^gama-1);
        end
        s(1:(order-j)) = tmpS((j+1):order);  %取对角值
    end
    yy(i) = tmpS(order);
end
format short;
```

例 15-17 通用外推法应用实例。用外推法求下面常微分方程的数值解。

$$\begin{cases} \dfrac{dy}{dx} = 1 + \cos x \quad 0 \leqslant x \leqslant 1 \\ y(0) = 0 \end{cases}$$

解： 在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = 1+cos(x);
>> yy = DEWT(z,0.1,0,1,3,0,4,[x y])
yy =    0
   0.1998
   0.3987
   0.5955
   0.7894
   0.9794
```

```
1.1646
1.3442
1.5174
1.6833
1.8415
```

将外推法的结果与理论解相比较：

x	外推法	理论解
0	0.0000	0.0000
0.1	0.1998	0.1998
0.2	0.3987	0.3987
0.3	0.5955	0.5955
0.4	0.7894	0.7894
0.5	0.9794	0.9794
0.6	1.1646	1.1646
0.7	1.3442	1.3442
0.8	1.5174	1.5174
0.9	1.6833	1.6833
1.0	1.8415	1.8415

从结果可以看出，外推法几乎是精确的。当然在此处的程序中，用的是二阶休恩法结合外推法得出的结果。

15.6.2 格拉格外推法

格拉格外推法给出了 $y(x_0+H;h_i)=p_i$ 的具体计算公式为：

$$\begin{cases} y_0 = y(x_0) \\ y_1 = y_0 + h_i f(x_0, y_0) \\ y_{n+2} = y_n + 2h_i f(x_{n+1}, y_{n+1}), n = 0,1,\cdots,N_i - 1 \\ y(x_0+H;h_i) = \dfrac{1}{4} y_{N_i-1} + \dfrac{1}{2} y_{N_i} + \dfrac{1}{4} y_{N_i+1} \end{cases}$$

在 MATLAB 中编程实现的格拉格外推法的函数为：yy = DEWT_glg(f,h,a,b,gama,y0,order,varvec)

功能：用格拉格外推法求一阶常微分方程的数值解

调用格式：yy = DEWT_glg(f,h,a,b,gama,y0,order,varvec)

其中，f：一阶常微分方程的一般表达式的右端函数；

h：积分步长；

a：自变量取值下限；

b：自变量取值上限；

gama：外推参数，参考外推公式；

y0：函数初值；
order：外推阶数；
varvec：常微分方程的变量组。

格拉格外推法的 MATLAB 程序代码如下所示：

```
function yy = DEWT_glg(f,h,a,b,gama,y0,order,varvec)
%一阶常微分方程的一般表达式的右端函数：f
%积分步长：h
%自变量取值下限：a
%自变量取值上限：b
%外推参数，参考外推公式：gama
%函数初值：y0
%外推阶数：order
%常微分方程的变量组：varvec
ArrayH = [1;2;4;6;8;12;16;24;32;48;64;96];
N = (b-a)/h;
yy = zeros(N+1,1);
for i = 2:N+1
    dh = h;
    s = zeros(order,1);
    for j=1:order
        dh = h/ArrayH(j);     %不同的 h 值
        NI = (i-1)* ArrayH(j);
        tmpY = zeros(NI+2,1);
        tmpY(1) = y0;
        tmpY(2) = y0 + dh*Funval(f,varvec,[a+dh y0]);
        for m=3:NI+1
            tmpY(m) = tmpY(m-2)+ 2*dh* Funval(f,varvec,[a+(m-1)*dh
tmpY(m-1)]);
        end                          %格拉格法得出的 y 初步值
        s(j) = tmpY(NI)/4+ tmpY(NI+1)/2+ tmpY(NI+2)/4;
    end
    tmpS = zeros(order,1);
    for j=1:order-1              %格拉格法外推公式
        for k=(j+1):order
            tmpS(k) = s(k)+(s(k)-s(k-1))/((ArrayH(k)/ArrayH(j))^gama-1);
        end
        s(1:(order-j)) = tmpS((j+1):order);
    end
    yy(i) = tmpS(order);
end
```

例 15-18 格拉格外推法应用实例。用格拉格外推法求下面常微分方程的数值解。

$$\begin{cases}\dfrac{dy}{dx}=1+\cos x \quad 0\leqslant x\leqslant 1\\ y(0)=0\end{cases}$$

解： 在 MATLAB 命令窗口中输入：

```
>> syms x y;
>> z = 1+cos(x);
>> yy = DEWT_glg(z,0.1,0,1,3,0,4,[x y])
yy =    0
     0.1997
     0.3983
     0.5947
     0.7880
     0.9773
     1.1616
     1.3402
     1.5122
     1.6769
     1.8337
```

此题的理论解为 $y = x + \sin x (0 \leqslant x \leqslant 1)$，数值解与理论解的比较结果如下：

x	理论解	格拉格数值解
0	0	0
0.1	0.1998	0.1997
0.2	0.3987	0.3983
0.3	0.5955	0.5947
0.4	0.7894	0.7880
0.5	0.9794	0.9773
0.6	1.1646	1.1616
0.7	1.3442	1.3402
0.8	1.5174	1.5122
0.9	1.6833	1.6769
1	1.8415	1.8337

随着 x 的增大，误差也不断增大，这是外推算法的固有特点。

15.7 小结

本章介绍了常见的求一阶常微分方程初值问题的方法，一阶常微分方程的解法是求解高阶常微分方程、常微分方程组以及边值问题的基础，因此熟练掌握这些方法，其他的问题也就很容易解决了。一般来说，高阶的龙格-库塔法是比较常用的方法，当计算量比较大时，采用预测-校正法也是不错的选择，而外推法结合各种显式法能给出非常高的精度。

第 16 章 偏微分方程的数值解法

偏微分方程是研究自然科学和社会科学中的事件、物体和现象运动、演化和变化规律的最为基本的数学方程。因此，偏微分方程数值解已用到科学技术和社会生活的各个领域中。

本章介绍了应用最为广泛的椭圆型、双曲型、抛物型偏微分方程的数值解法，而且还详细编程实现了每种方程的多种常见数值解法。

通过本章的学习，读者不仅能掌握常见偏微分方程的多种数值求解法，而且还能熟练使用 MATLAB 编程来实现这些算法。

16.1 椭圆偏微分方程

拉普拉斯方程是最简单的椭圆偏微分方程，下面以拉普拉斯方程为例，讲述椭圆偏微分方程的数值解法。拉普拉斯方程的形式如下所示：

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=0$$

椭圆偏微分方程的边界条件有以下三种提法。

- 固定边界条件：$u|_\Gamma=U_1(x,y)$，即在边界Γ上给定u的值$U_1(x,y)$。
- 给定法向导数的边界条件：$\frac{\partial u}{\partial n}|_\Gamma=U_2(x,y)$，即在边界$\Gamma$上给定$u$的法向导数值$U_2(x,y)$。
- 混合边界条件：$(\frac{\partial u}{\partial n}+ku)|_\Gamma=U_3(x,y)$，即在边界$\Gamma$上，$u$和$u$的法向导数值加起来等于$U_3(x,y)$。

其中第一种边界条件的提法最为普遍，下面以第一种边界条件为例，介绍椭圆偏微分方程常用的五点差分格式和工字型差分格式的解法。

16.1.1 五点差分格式

五点差分格式是最常用的格式，其形式如下所示：

$$u_{i+1,j}+u_{i-1,j}+u_{i,j+1}+u_{i,j-1}=4u_{i,j}$$

上式涉及的格点如图 16-1 所示：

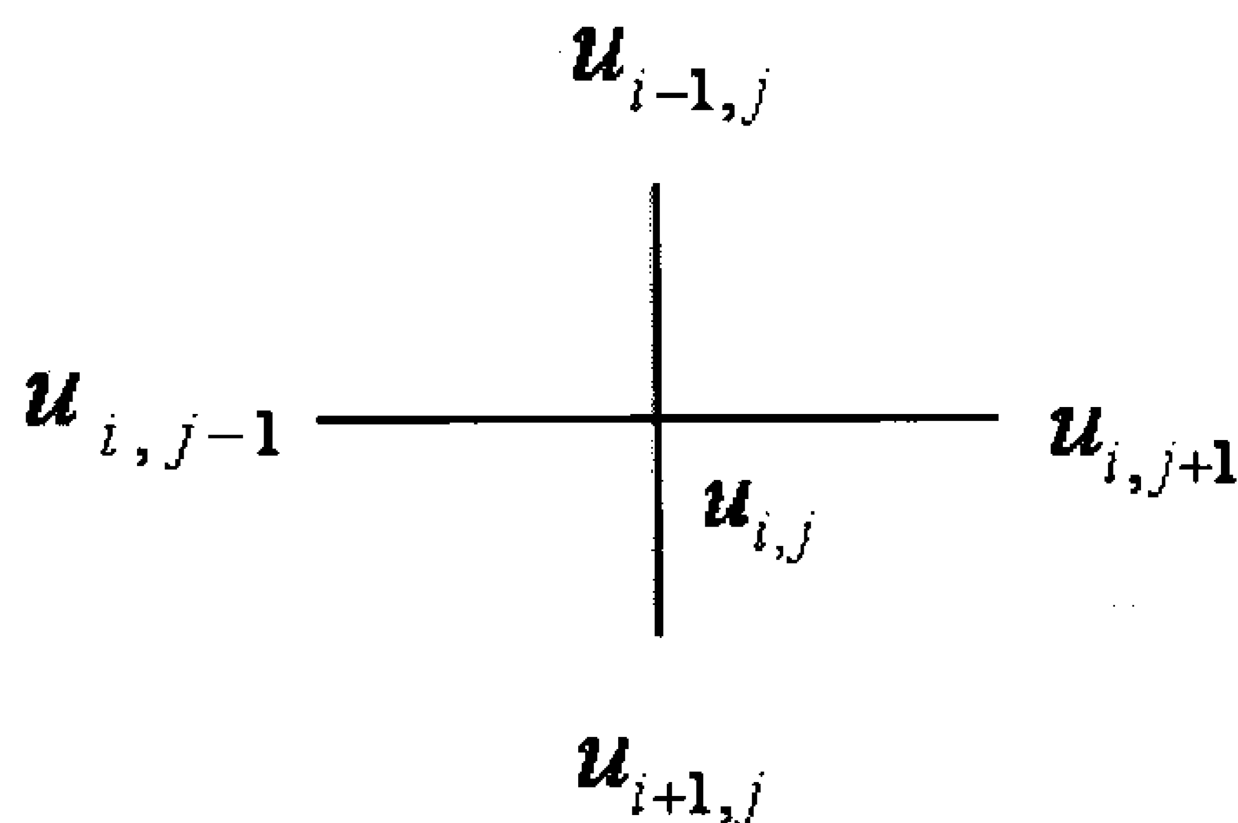


图 16-1 五点差分格式的格点图

图 16-1 是将方程求解域用格点离散后取的相邻五个格点，这五个格点处的函数值满足差分格式 $u_{i+1,j}+u_{i-1,j}+u_{i,j+1}+u_{i,j-1}=4u_{i,j}$。

五点差分格式用来求下列边值问题：

$$\begin{cases}\dfrac{\partial^2 u}{\partial x^2}+\dfrac{\partial^2 u}{\partial y^2}=0\\ u(x_1,y)=g_1(x),u(x_2,y)=g_2(x)\\ u(x,y_1)=f_1(y),u(x,y_2)=f_2(y)\\ x_1\leqslant x\leqslant x_2,y_1\leqslant y\leqslant y_2\end{cases}$$

其中 $g_1(x)$ 和 $g_2(x)$ 是关于 x 的函数， $f_1(y)$ 和 $f_2(y)$ 是关于 y 的函数。

用五点差分格式求解拉普拉斯方程的算法过程介绍如下。

❶ 对求解区域进行分割：将 $x_{\min}\leqslant x\leqslant x_{\max}$ 范围内的 x 轴等分为 NX 段，将 $y_{\min}\leqslant y\leqslant y_{\max}$ 范围内的 y 轴等分为 NY 段；

❷ 将边界条件离散到格点上；

❸ 用五点差分格式建立求解方程，求出各个格点的函数值。



要保证 x 方向和 y 方向上的网格步长相等才可使用上面的公式。

在 MATLAB 中编程实现的五点差分格式的函数为：peEllip5

功能：用五点差分格式解拉普拉斯方程

调用格式：u = peEllip5(nx,minx,maxx,ny,miny,maxy)

其中，nx：x 方向的节点数；

minx：求解区间 x 的左端；

maxx：求解区间 x 的右端；

ny：y 方向的节点数；

minx：求解区间 y 的左端；

maxx：求解区间 y 的右端；

u：求解区间上的数值解。

五点差分格式解拉普拉斯方程边值问题的 MATLAB 程序代码如下所示：

```
function u = peEllip5(nx,minx,maxx,ny,miny,maxy)
% x 方向的节点数: nx
%求解区间 x 的左端: minx
%求解区间 x 的右端: maxx
% y 方向的节点数: ny
%求解区间 y 的左端: miny
%求解区间 y 的右端: maxy
%求解区间上的数值解: u
format long;
hx = (maxx-minx)/(nx-1);
hy = (maxy-miny)/(ny-1);
u0 = zeros(nx,ny);
for j=1:ny
    u0(j,1) = EllIni2Uxl(minx,miny+(j-1)*hy);
    u0(j,nx) = EllIni2Uxr(maxx,miny+(j-1)*hy);
end
for j=1:nx
    u0(1,j) = EllIni2Uyl(minx+(j-1)*hx,miny);
    u0(ny,j) = EllIni2Uyr(minx+(j-1)*hx,maxy);     %边界条件的离散
end
A = -4*eye((nx-2)*(ny-2),(nx-2)*(ny-2));
b = zeros((nx-2)*(ny-2),1);
for i=1:(nx-2)*(ny-2)
    if mod(i,nx-2) == 1                        %注意离边界点最近节点的离散方式
        if i==1
            A(1,2) = 1;
            A(1,nx-1) = 1;
            b(1) = - u0(1,2) - u0(2,1);
        else
            if i == (ny-3)*(nx-2)+1
                A(i,i+1) = 1;
                A(i,i-nx+2) = 1;
                b(i) = - u0(ny-1,1) - u0(ny,2);
            else
                A(i,i+1) = 1;
                A(i,i-nx+2) = 1;
                A(i,i+nx-2) = 1;
                b(i) = - u0(floor(i/(nx-2))+2,1);
            end
        end
    else
        if mod(i,nx-2) == 0                    %注意离边界点最近节点的离散方式
            if i == nx-2
                A(i,i-1) = 1;
                A(i,i+nx-2) = 1;
                b(i) = - u0(1,nx-1) - u0(2,nx);
            else
                if i == (ny-2)*(nx-2)
                    A(i,i-1) = 1;
                    A(i,i-nx+2) = 1;
```

```
                b(i) = - u0(ny-1,nx) - u0(ny,nx-1);
            else
                A(i,i-1) = 1;
                A(i,i-nx+2) = 1;
                A(i,i+nx-2) = 1;
                b(i) = - u0(floor(i/(nx-2))+1,nx);
            end
        end
    else
        if i>1 && i< nx-2
            A(i,i-1) = 1;
            A(i,i+nx-2) = 1;
            A(i,i+1) = 1;
            b(i) = - u0(1,i+1);
        else
            if i > (ny-3)*(nx-2) && i < (ny-2)*(nx-2)
                A(i,i-1) = 1;
                A(i,i-nx+2) = 1;
                A(i,i+1) = 1;
                b(i) = - u0(ny,mod(i,(nx-2))+1);
            else                                    %其余节点的离散方式
                A(i,i-1) = 1;
                A(i,i+1) = 1;
                A(i,i+nx-2) = 1;
                A(i,i-nx+2) = 1;
            end
        end
    end
  end
end
u = A\b;
format short;
```

例 16-1 五点差分格式求解拉普拉斯方程边值问题应用实例。用五点差分格式求解下面拉普拉斯方程的边值问题：

$$\begin{cases} \dfrac{\partial^2 u}{\partial x^2}+\dfrac{\partial^2 u}{\partial y^2}=0 & 0\leqslant x\leqslant 2, 0\leqslant y\leqslant 2 \\ u(0,y)=0 & u(2,y)=y(2-y) \\ u(x,0)=0 & u(x,2)=\begin{cases} x & x<1 \\ 2-x & x>1 \end{cases} \end{cases}$$

其中空间步长都取 0.04。

解：先建立 4 个 M 文件，以建立边界条件：

```
function uxy = EllIni2Uxl(x,y)
format long;
uxy = 0;
function uxy = EllIni2Uxr(x,y)
```

```
format long;
uxy = y*(2-y);
function uxy = EllIni2Uyl(x,y)
format long;
uxy = 0;
function uxy = EllIni2Uyr(x,y)
format long;
if x < 1
   uxy = x;
else
   uxy = 2 - x;
end
```

然后在 MATLAB 窗口输入下列命令:

```
u = peEllip5(51,0,2,51,0,2);
```

结果如图 16-2 所示:

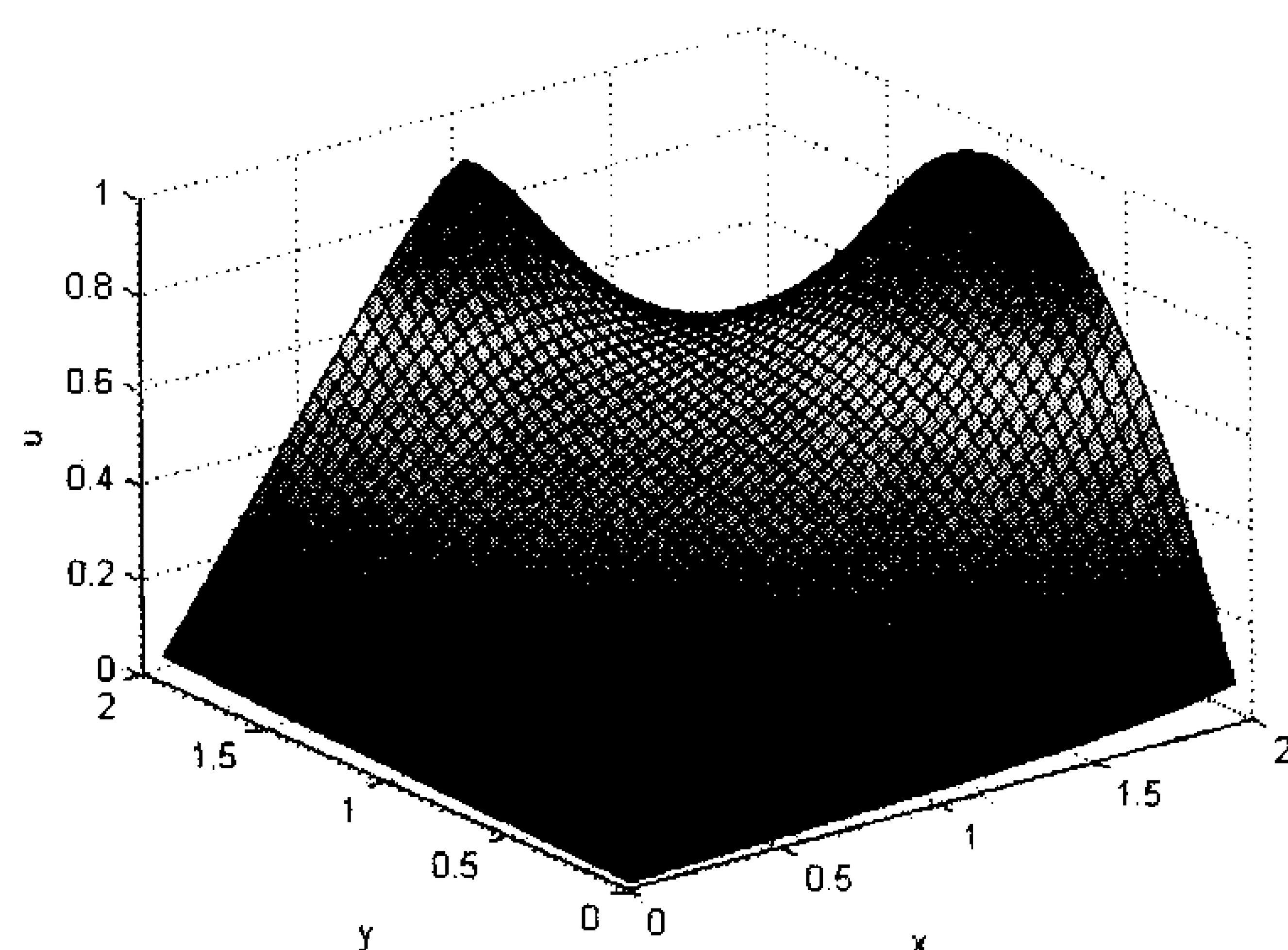


图 16-2　例 16-1 中方程在求解域上的场函数值图

在图 16-2 中，不同的颜色代表不同的 u 值，如果网格更密的话，即采取更多的节点进行计算的话，会得到更加光滑的曲面。

16.1.2　工字型差分格式

工字型差分格式的形式如下所示:

$$u_{i+1,j+1}+u_{i-1,j-1}+u_{i-1,j+1}+u_{i+1,j-1}=4u_{i,j}$$

上式涉及的格点如图 16-3 所示:

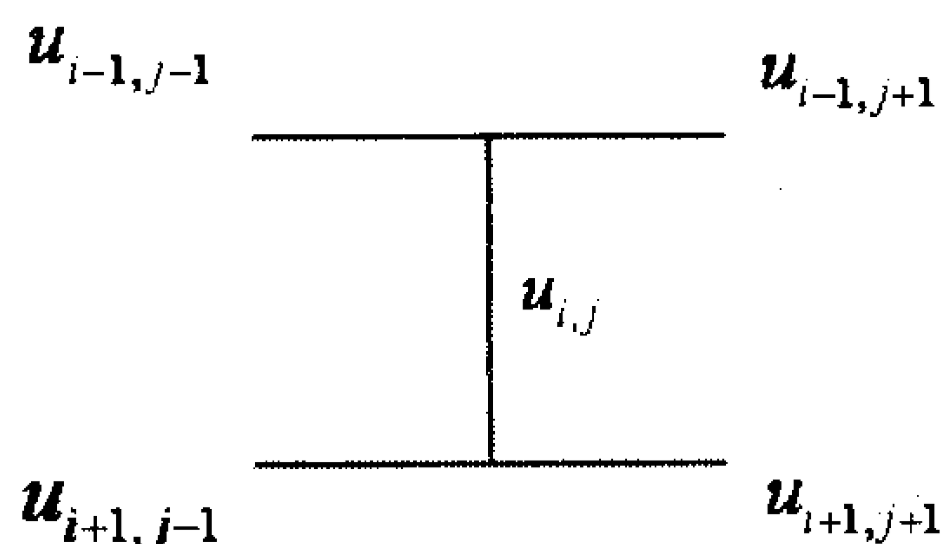


图 16-3　工字型差分格式的格点图

上图是将方程求解域用格点离散后取的相邻五个格点，这五个格点处的函数值满足差分格式 $u_{i+1,j+1}+u_{i-1,j-1}+u_{i-1,j+1}+u_{i+1,j-1}=4u_{i,j}$。

工字型差分格式用来求下列边值问题：

$$\begin{cases}\dfrac{\partial^2 u}{\partial x^2}+\dfrac{\partial^2 u}{\partial y^2}=0\\ u(x_1,y)=g_1(x),u(x_2,y)=g_2(x)\\ u(x,y_1)=f_1(y),u(x,y_2)=f_2(y)\\ x_1\leqslant x\leqslant x_2,y_1\leqslant y\leqslant y_2\end{cases}$$

其中 $g_1(x)$ 和 $g_2(x)$ 是关于 x 的函数，$f_1(y)$ 和 $f_2(y)$ 是关于 y 的函数。

用工字型差分格式求解拉普拉斯方程的算法过程介绍如下。

❶ 对求解区域进行分割：将 $x_{\min}\leqslant x\leqslant x_{\max}$ 范围内的 x 轴等分为 NX 段，将 $y_{\min}\leqslant y\leqslant y_{\max}$ 范围内的 y 轴等分为 NY 段；

❷ 将边界条件离散到格点上；

❸ 用工字型差分格式建立求解方程，求出各个格点的函数值。



要保证 x 方向和 y 方向上的网格步长相等才可使用上面的公式。

在 MATLAB 中编程实现的工字型差分格式的函数为：peEllip5m

功能：用工字型差分格式解拉普拉斯方程

调用格式：u = peEllip5m(nx,minx,maxx,ny,miny,maxy)

其中，nx：x 方向的节点数；
minx：求解区间 x 的左端；
maxx：求解区间 x 的右端；
ny：y 方向的节点数；
minx：求解区间 y 的左端；
maxx：求解区间 y 的右端；
u：求解区间上的数值解。

工字型差分格式求解拉普拉斯方程边值问题的 MATLAB 程序代码如下所示：

```
function u = peEllip5m(nx,minx,maxx,ny,miny,maxy)
% x 方向的节点数：nx
%求解区间 x 的左端：minx
%求解区间 x 的右端：maxx
% y 方向的节点数：ny
%求解区间 y 的左端：miny
%求解区间 y 的右端：maxy
%求解区间上的数值解：u
format long;
hx = (maxx-minx)/(nx-1);
hy = (maxy-miny)/(ny-1);
```

```
    u0 = zeros(nx,ny);
    for j=1:ny
        u0(j,1) = EllIni2Uxl(minx,miny+(j-1)*hy);
        u0(j,nx) = EllIni2Uxr(maxx,miny+(j-1)*hy);
    end
    for j=1:nx
        u0(1,j) = EllIni2Uyl(minx+(j-1)*hx,miny);
        u0(ny,j) = EllIni2Uyr(minx+(j-1)*hx,maxy);     %边界条件的离散
    end
    A = -4*eye((nx-2)*(ny-2),(nx-2)*(ny-2));
    b = zeros((nx-2)*(ny-2),1);
    for i=1:(nx-2)*(ny-2)
        if mod(i,nx-2) == 1                          %注意离边界点最近节点的离散方式
            if i==1
                A(1,nx) = 1;
                b(1) = - u0(1,1) - u0(3,1) - u0(1,3);
            else
                if i == (ny-3)*(nx-2)+1
                    A(i,i-nx+1) = 1;
                    b(i) = - u0(ny,1) - u0(ny,3) - u0(ny-2,1);
                else
                    A(i,i-nx+3) = 1;
                    A(i,i+nx-1) = 1;
                    b(i) = - u0(floor(i/(nx-2))+1,1) - u0(floor(i/(nx-2))+3,1);
                end
            end
        else
            if mod(i,nx-2) == 0                      %注意离边界点最近节点的离散方式
                if i == nx-2
                    A(i,i+nx-1) = 1;
                    b(i) = - u0(1,nx-2) - u0(1,nx) - u0(3,nx);
                else
                    if i == (ny-2)*(nx-2)
                        A(i,i-nx+1) = 1;
                        b(i) = - u0(ny,nx) - u0(ny,nx-2) - u0(ny-2,nx);
                    else
                        A(i,i-nx+1) = 1;
                        A(i,i+nx-3) = 1;
                        b(i) = - u0(floor(i/(nx-2)),nx) -
u0(floor(i/(nx-2))+2,nx);
                    end
                end
            else
                if i>1 && i< nx-2
                    A(i,i+nx-1) = 1;
                    A(i,i+nx-3) = 1;
                    b(i) = - u0(1,i) - u0(1,i+2);
                else
                    if i > (ny-3)*(nx-2) && i < (ny-2)*(nx-2)
                        A(i,i-nx+3) = 1;
                        A(i,i-nx+1) = 1;
```

```
                    b(i) = - u0(ny,mod(i,(nx-2))) -u0(ny,mod(i,(nx-2))+2);
                else
                    A(i,i-nx+3) = 1;          %其余节点的离散方式
                    A(i,i+nx-1) = 1;
                    A(i,i+nx-3) = 1;
                    A(i,i-nx+1) = 1;
                end
            end
        end
    end
end
ul = A\b;
for i=1:(ny-2)
    for j=1:(nx-2)
        u(i,j) = ul((i-1)*(nx-2)+j);
    end
end
format short;
```

例 16-2 工字型差分格式求解拉普拉斯方程边值问题应用实例。用工字型差分格式求解下面拉普拉斯方程的边值问题：

$$\begin{cases} \dfrac{\partial^2 u}{\partial x^2}+\dfrac{\partial^2 u}{\partial y^2}=0 \quad 0\leqslant x\leqslant 2, 0\leqslant y\leqslant 2 \\ u(0,y)=0, u(2,y)=y(2-y) \\ u(x,0)=0 \\ u(x,2)=\begin{cases} x & x<1 \\ 2-x & x>1 \end{cases} \end{cases}$$

其中空间步长都取 0.04。

解：先建立 4 个 M 文件，以建立边界条件：

```
function uxy = EllIni2Uxl(x,y)
format long;
uxy = 0;
function uxy = EllIni2Uxr(x,y)
format long;
uxy = y*(2-y);
function uxy = EllIni2Uyl(x,y)
format long;
uxy = 0;
function uxy = EllIni2Uyr(x,y)
format long;
if x < 1
    uxy = x;
else
    uxy = 2 - x;
end
```

然后在 MATLAB 窗口输入下列命令：

```
u = peEllip5(51,0,2,51,0,2);
```

结果如图 16-4 所示：

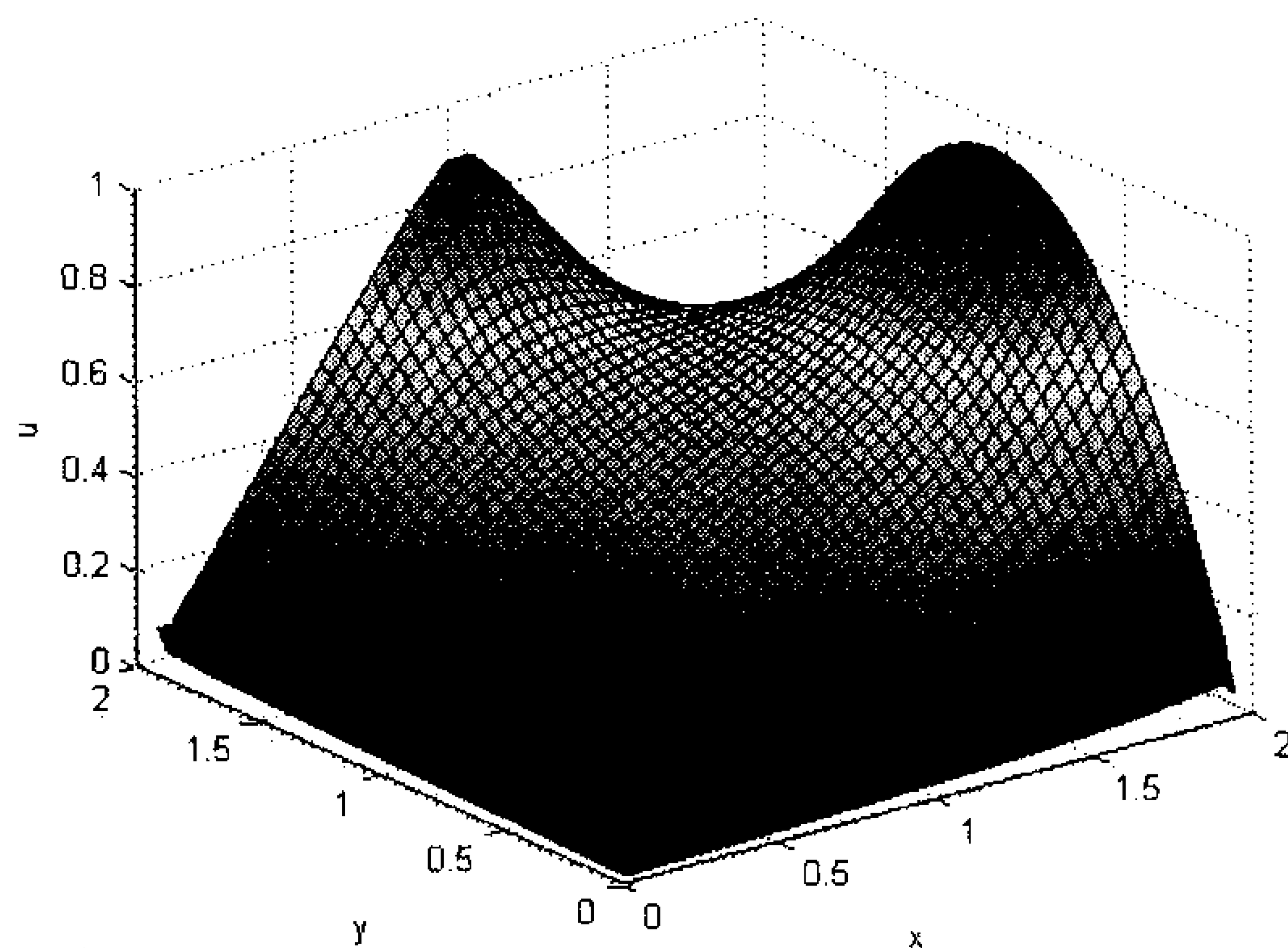


图 16-4　例 16-2 中方程在求解域上的场函数值图

用工字型差分格式算出的结果和五点差分格式得到的结果差不多，但是前者在角点上算得不太好，这是格式自身的缺点。

16.2 双曲线偏微分方程

对流方程是最简单的双曲线偏微分方程，下面以一维、二维对流方程为例，讲述对流方程的数值解法。

对流方程研究得都比较透彻，而且其差分格式是解决各种复杂双曲线偏微分方程的基础。

16.2.1 一维对流方程

一维对流方程的形式如下所示：

$$\frac{\partial u}{\partial t}+a\frac{\partial u}{\partial x}=0, x\in(-\infty,\infty), t>0, a \text{ 为常数}$$

如果给定初始条件：

$$u(x,0)=U(x), x\in(-\infty,\infty)$$

则一维对流方程的通解为：

$$u(x,t)=U(x-at), x\in(-\infty,\infty), t>0$$

一维对流方程形式简单，其差分格式非常多，常见的有迎风格式、拉克斯-弗里德里希斯格式、拉克斯-温德洛夫格式、比姆-沃明格式、Richtmyer 多步格式、拉克斯-温德洛夫多步格式和 MacCormack 多步格式，下面分别进行讲述。

16.2.1.1 迎风格式

迎风格式的形式如下所示：

$$\frac{u_j^{n+1}-u_j^n}{\tau}+\frac{a}{h}(u_j^n-u_{j-1}^n)=0 \quad a>0$$

$$\frac{u_j^{n+1}-u_j^n}{\tau}+\frac{a}{h}(u_{j+1}^n-u_j^n)=0 \quad a<0$$

其中τ为时间步长，h为空间步长。

以$a>0$为例，当用迎风格式求解对流方程时，在计算求解区间的左端点处的下一个时刻的函数值时，要用到左端点的左边一个节点的值，因此必须向左延拓一个节点，才能计算下一个时刻的左端点的函数值，如此得出，M 个时间步的迎风格式，应向左延拓 M 个节点函数值，这点在程序中可以看出。

> 在 MATLAB 中编程实现的迎风格式的函数为：peHypbYF
>
> 功能：用迎风格式解对流方程
>
> 调用格式：u = peHypbYF (a,dt,n,minx,maxx,M)
>
> 其中，a：方程中的常数；
>
> dt：时间步长；
>
> n：空间节点个数；
>
> minx：求解区间的左端；
>
> maxx：求解区间的右端；
>
> M：时间步的个数；
>
> u：求解区间上的数值解。

迎风格式的 MATLAB 程序代码如下所示：

```
function u = peHypbYF (a,dt,n,minx,maxx,M)
%方程中的常数：a
%时间步长：dt
%空间节点个数：n
%求解区间的左端：minx
%求解区间的右端：maxx
%时间步的个数：M
%求解区间上的数值解：u
format long;
h = (maxx-minx)/(n-1);
if a>0
    for j=1:(n+M)
        u0(j)  = IniU(minx+(j-M-1)*h);     %向左延拓 M个节点的函数值
    end
else
    for j=1:(n+M)
        u0(j)  = IniU(minx+(j-1)*h);      %向左延拓 M个节点的函数值
```

```
    end
end
u1 = u0;
for k=1:M
    if a>0
        for i=(k+1):n+M
            u1(i) = -dt*a*(u0(i)-u0(i-1))/h+u0(i);
        end
    else
        for i=1:n+M-k
            u1(i) = -dt*a*(u0(i+1)-u0(i))/h+u0(i);
        end
    end
    u0 = u1;
end
if a>0
    u = u1((M+1):M+n);
else
    u = u1(1:n);
end
format long;
```

例 16-3 迎风格式求解一维对流方程应用实例。用迎风格式求解下面的初值问题：

$$\begin{cases} \dfrac{\partial u}{\partial t}+\dfrac{\partial u}{\partial x}=0 & x\in(-\infty,\infty),t>0 \\ u(x,0)=U(x) & x\in(-\infty,\infty) \end{cases}$$

其中时间步长取 0.005，求解区间为[0,1]，空间步长取 0.01，求出当 t=0.5（即 100 个时间步）时的 u 随 x 的分布图。

$$U(x)=\begin{cases} 10x+1 & -0.1\leqslant x\leqslant 0 \\ -10x+1 & 0\leqslant x\leqslant 0.1 \\ 0 & \text{其余} \end{cases}$$

解：先建立一个名为 IniU.m 的 MATLAB 文件，输入如下内容：

```
function ux = IniU(x)
format long;
if x<=0
    if x >= -0.1
        ux = 10*(x+0.1);
    else
        ux = 0;
    end
else
    if x <= 0.1
        ux = -10*(x-0.1);
    else
        ux = 0;
    end
end
```

然后在 MATLAB 窗口输入下列命令：

```
>> u = peHypbYF(1,0.005,101,0,1,100);
```

将初值（即 t=0 时）与用迎风格式求的结果用图表示，如图 16-5 及图 16-6 所示。

当 t=0.5 时，可以看出开始时的函数形状到这时已经有所变化，高度有所降低，而宽度却增大了。

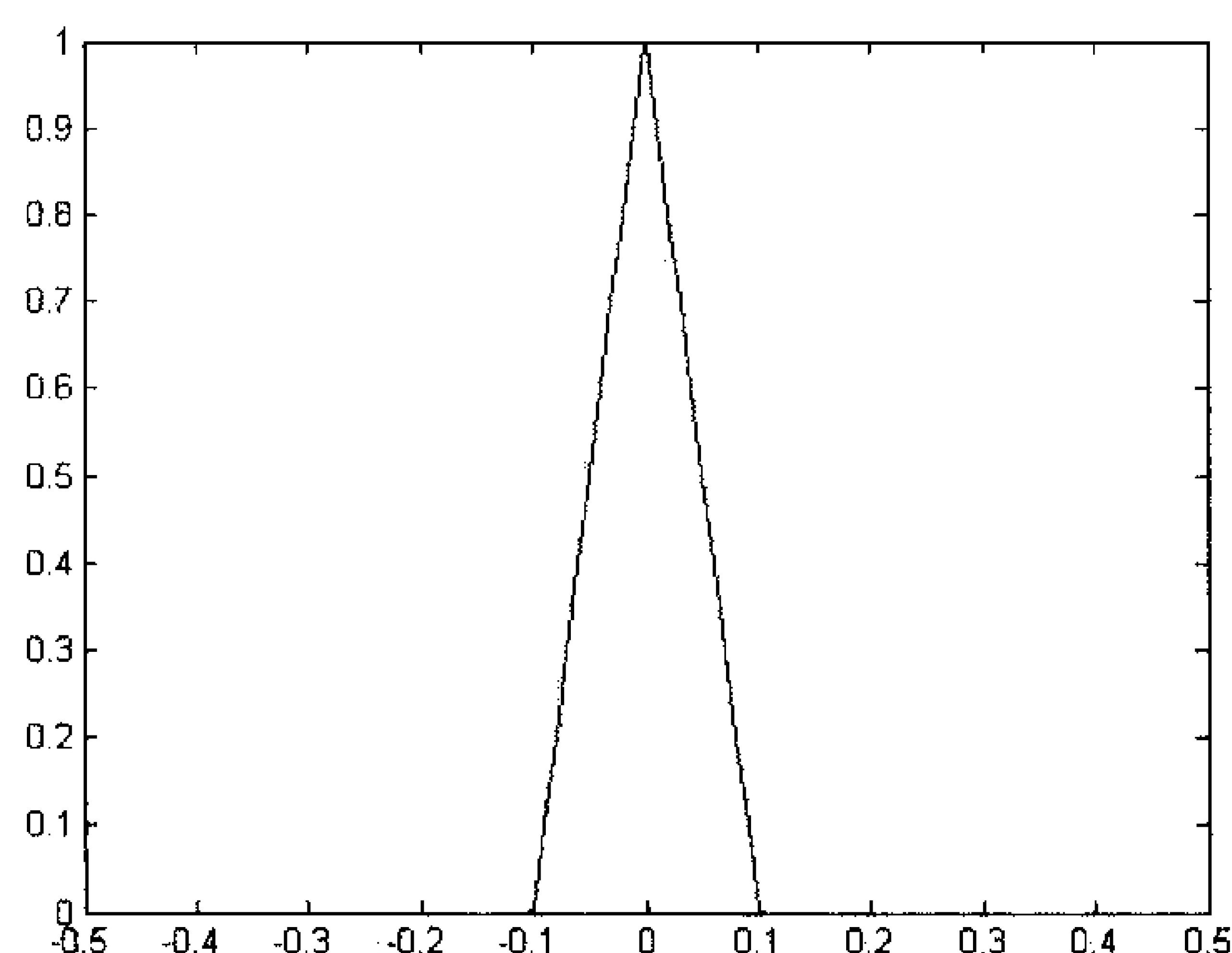


图 16-5　t=0 时的 u 值

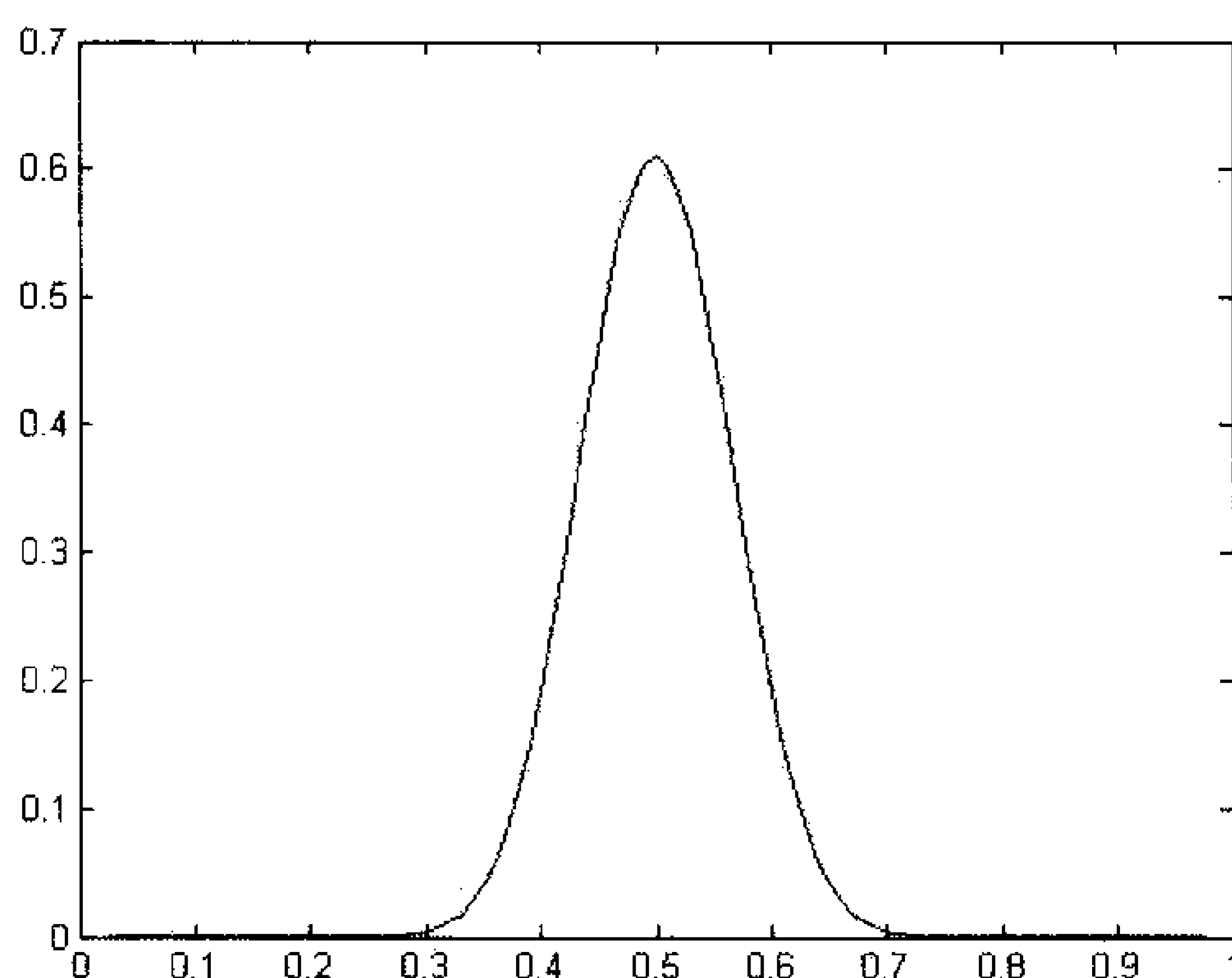


图 16-6　t=0.5 时的 u 值

16.2.1.2　拉克斯-弗里德里希斯格式

拉克斯-弗里德里希斯格式的形式如下所示：

$$\frac{u_j^{n+1}-\frac{1}{2}(u_{j+1}^n+u_{j-1}^n)}{\tau}+\frac{a}{2h}(u_{j+1}^n-u_{j-1}^n)=0$$

其中 τ 为时间步长，h 为空间步长。

当用拉克斯-弗里德里希斯格式求解对流方程时，在计算求解区间左端点和右端点处下一个时刻的函数值时，分别要用到左端点左边一个节点的值和右端点右边一个节点的值，因此每个时间步都必须向左延拓一个节点，向右延拓一个节点，才能分别计算下一个

时刻左端点的函数值和右端点的函数值，如此得出，M 个时间步的拉克斯-弗里德里希斯格式，需分别向左延拓 M 个节点函数值，向右延拓 M 个节点函数值，这点在程序中可以看出。

在 MATLAB 中编程实现的拉克斯-弗里德里希斯格式的函数为：peHypbLax

功能：用拉克斯-弗里德里希斯格式解对流方程

调用格式：u = peHypbLax (a,dt,n,minx,maxx,M)

其中，a：方程中的常数；

dt：时间步长；

n：空间节点个数；

minx：求解区间的左端；

maxx：求解区间的右端；

M：时间步的个数；

u：求解区间上的数值解。

拉克斯-弗里德里希斯格式的 MATLAB 程序代码如下所示：

```
function u = peHypbLax (a,dt,n,minx,maxx,M)
%方程中的常数：a
%时间步长：dt
%空间节点个数：n
%求解区间的左端：minx
%求解区间的右端：maxx
%时间步的个数：M
%求解区间上的数值解：u
format long;
h = (maxx-minx)/(n-1);
for j=1:(n+2*M)
        u0(j) = IniU(minx+(j-M-1)*h);    %左右分别各延拓 M 个节点的函数值
end
u1 = u0;
for k=1:M
    for i=k+1:n+2*M-k
        u1(i) = -dt*a*(u0(i+1)-u0(i-1))/h/2+(u0(i+1)+u0(i-1))/2;
    end
    u0 = u1;
end
u = u1((M+1):(M+n));
format short;
```

例 16-4 拉克斯-弗里德里希斯格式求解一维对流方程应用实例。用拉克斯-弗里德里希斯格式求解下面的初值问题：

$$\begin{cases} \dfrac{\partial u}{\partial t}+\dfrac{\partial u}{\partial x}=0 & x\in(-\infty,\infty),t>0 \\ u(x,0)=U(x) & x\in(-\infty,\infty) \end{cases}$$

其中时间步长取 0.005，求解区间为[0,1]，空间步长取 0.01，求出当 t=0.5（即 100 个时间步）时的 u 随 x 的分布图。

$$U(x)=\begin{cases}10x+1 & -0.1\leqslant x\leqslant 0\\ -10x+1 & 0\leqslant x\leqslant 0.1\\ 0 & \text{其余}\end{cases}$$

解：先建立一个名为 IniU.m 的 MATLAB 文件，输入如下内容：

```
function ux = IniU(x)
format long;
if x<=0
    if x >= -0.1
        ux = 10*(x+0.1);
    else
        ux = 0;
    end
else
    if x <= 0.1
        ux = -10*(x-0.1);
    else
        ux = 0;
    end
end
```

然后在 MATLAB 窗口输入下列命令：

```
>> u = peHypbLax (1,0.005,101,0,1,100);
```

用拉克斯-弗里德里希斯格式求的结果如图 16-7 所示：

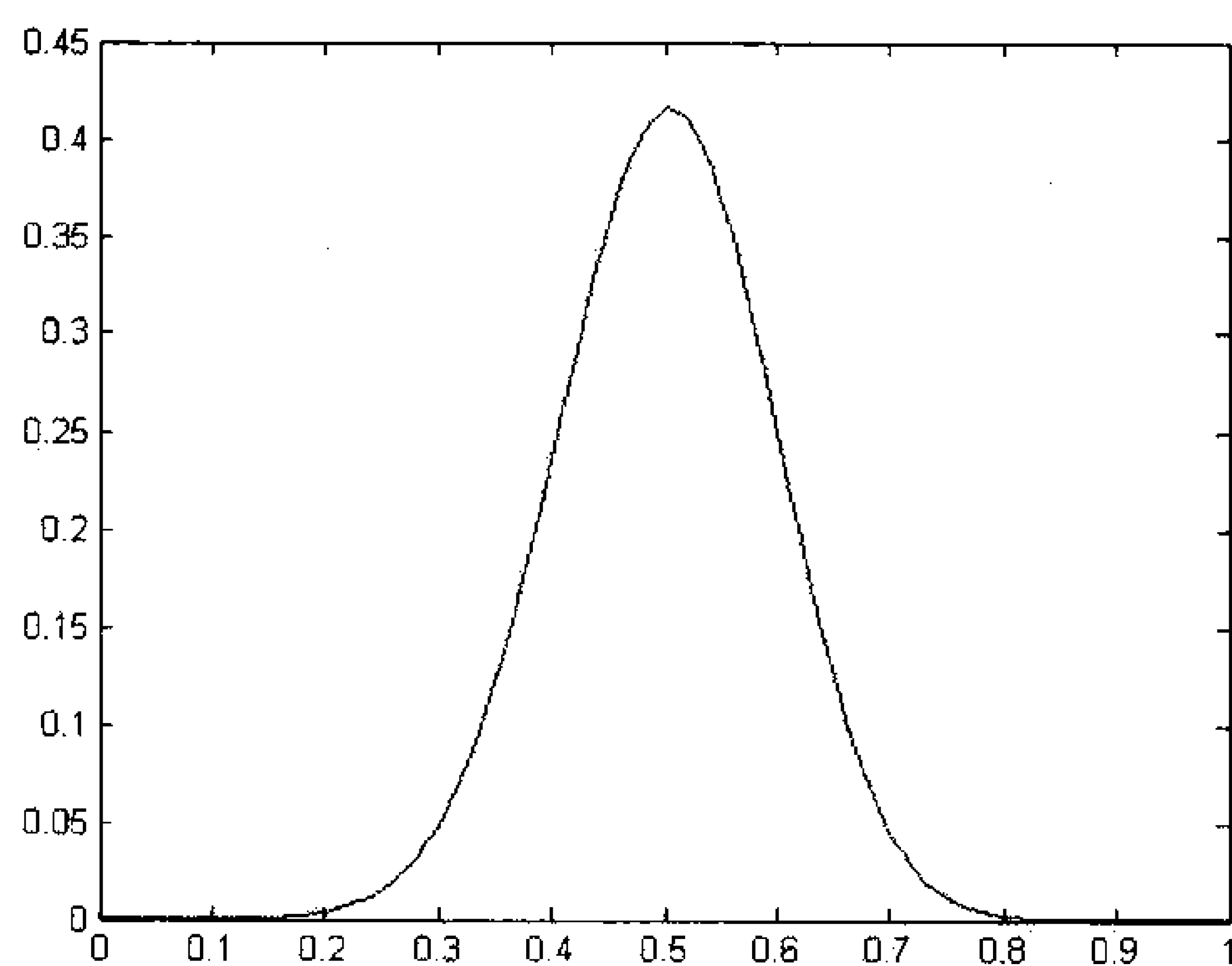


图 16-7 用拉克斯-弗里德里希斯格式求得的当 t=0.5 时的 u 值

从图 16-7 可以看出，用拉克斯-弗里德里希斯格式求得结果的高度比迎风格式结果更低，宽度也更大。

16.2.1.3 拉克斯-温德洛夫格式

拉克斯-温德洛夫格式的形式如下所示：

$$\frac{u_j^{n+1}-u_j^n}{\tau}+\frac{a}{2h}(u_{j+1}^n-u_{j-1}^n)-\frac{\tau a^2}{2h}(u_{j+1}^n-2u_j^n+u_{j-1}^n)=0$$

其中τ为时间步长，h为空间步长。

同拉克斯-弗里德里希斯格式一样，拉克斯-温德洛夫格式也需要进行延拓。

在 MATLAB 中编程实现的拉克斯-温德洛夫格式的函数为：peHypbLaxW

功能：用拉克斯-温德洛夫格式解对流方程

调用格式：u = peHypbLaxW (a,dt,n,minx,maxx,M)

其中，a：方程中的常数；

dt：时间步长；

n：空间节点个数；

minx：求解区间的左端；

maxx：求解区间的右端；

M：时间步的个数；

u：求解区间上的数值解。

拉克斯-温德洛夫格式的 MATLAB 程序代码如下所示：

```
function u = peHypbLaxW (a,dt,n,minx,maxx,M)
%方程中的常数：a
%时间步长：dt
%空间节点个数：n
%求解区间的左端：minx
%求解区间的右端：maxx
%时间步的个数：M
%求解区间上的数值解：u
format long;
h = (maxx-minx)/(n-1);
for j=1:(n+2*M)
        u0(j) = IniU(minx+(j-M-1)*h);          %节点延拓
end
u1 = u0;
for k=1:M
    for i=k+1:n+2*M-k
        u1(i) = dt*dt*a*a*(u0(i+1)-2*u0(i)+u0(i-1))/2/h/h - ...
                   dt*a*(u0(i+1)-u0(i-1))/h/2+u0(i);
    end
    u0 = u1;
end
u = u1((M+1):(M+n));
format short;
```

例 16-5 拉克斯-温德洛夫格式求解一维对流方程应用实例。用拉克斯-温德洛夫格式求解下面的初值问题：

$$\begin{cases}\dfrac{\partial u}{\partial t}+\dfrac{\partial u}{\partial x}=0 & x\in(-\infty,\infty),t>0\\ u(x,0)=U(x) & x\in(-\infty,\infty)\end{cases}$$

其中时间步长取 0.005，求解区间为[0,1]，空间步长取 0.01，求出当 t=0.5（即 100 个时间步）时的 u 随 x 的分布图。

$$U(x)=\begin{cases}10x+1 & -0.1\leqslant x\leqslant 0\\ -10x+1 & 0\leqslant x\leqslant 0.1\\ 0 & 其余\end{cases}$$

解：先建立一个名为 IniU.m 的 MATLAB 文件，输入如下内容：

```
function ux = IniU(x)
format long;
if x<=0
    if x >= -0.1
        ux = 10*(x+0.1);
    else
        ux = 0;
    end
else
    if x <= 0.1
        ux = -10*(x-0.1);
    else
        ux = 0;
    end
end
```

然后在 MATLAB 窗口输入下列命令：

```
>> u = peHypbLaxW (1,0.005,101,0,1,100);
```

用拉克斯-温德洛夫格式求的结果如图 16-8 所示：

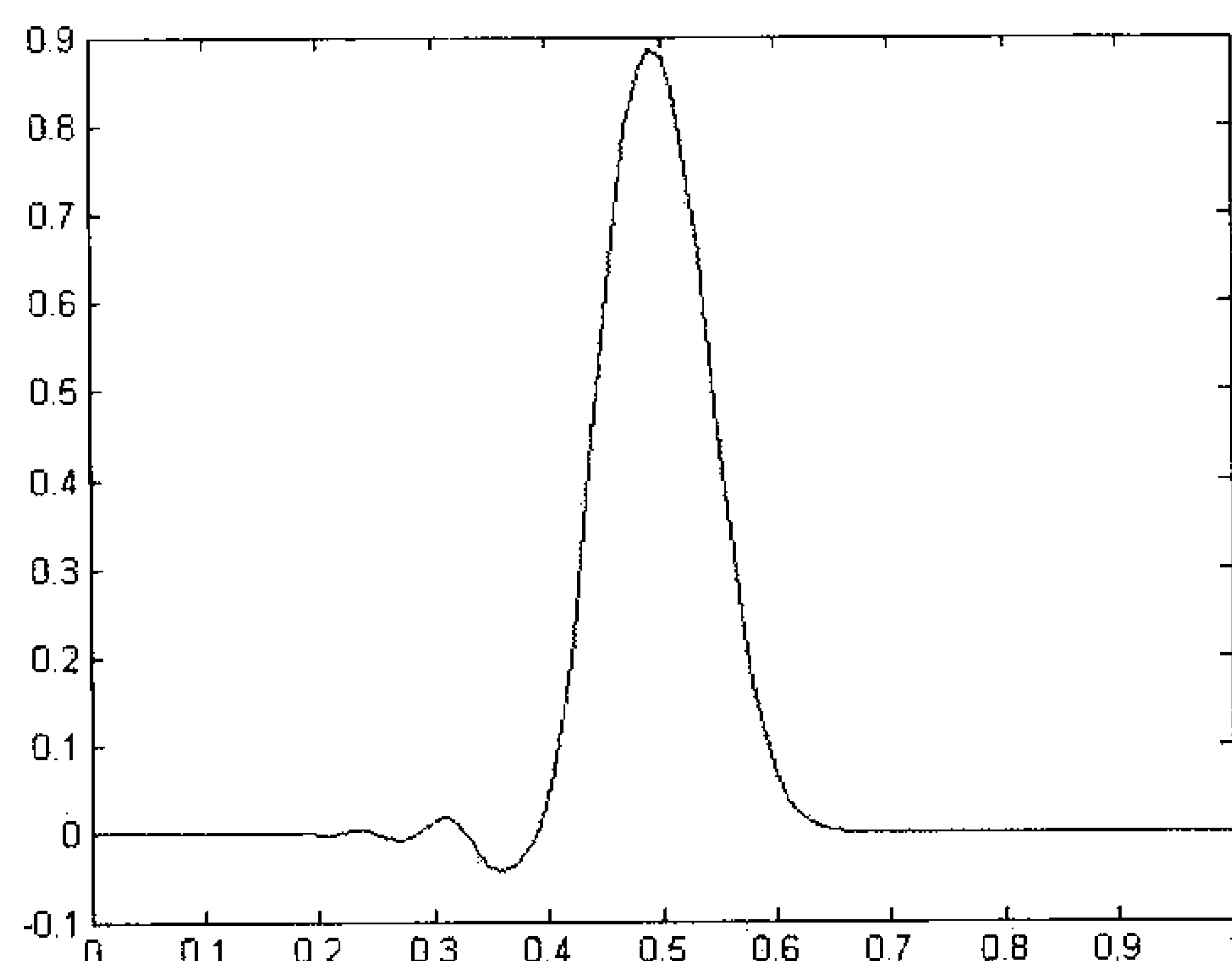


图 16-8 用拉克斯-温德洛夫格式求得的当 t=0.5 时的 u 值

从图 16-8 可以明显地看出，拉克斯-温德洛夫格式的特点是函数值在左边有波动。

16.2.1.4 比姆-沃明格式

比姆-沃明格式的形式如下所示：

$$\frac{u_j^{n+1}-u_j^n}{\tau}+\frac{a}{h}(u_j^n-u_{j-1}^n)+\frac{a}{2h}(1-\frac{a\tau}{h})(u_j^n-2u_{j-1}^n+u_{j-2}^n)=0$$

其中τ为时间步长，h为空间步长。

同理，比姆-沃明格式也需要拓展 2M 个节点的函数值，但它的节点全是向左延拓的。

> 在 MATLAB 中编程实现的比姆-沃明格式的函数为：peHypbBW
>
> 功能：用比姆-沃明格式解对流方程
>
> 调用格式：u = peHypbBW (a,dt,n,minx,maxx,M)
>
> 其中，a：方程中的常数；
>
> dt：时间步长；
>
> n：空间节点个数；
>
> minx：求解区间的左端；
>
> maxx：求解区间的右端；
>
> M：时间步的个数；
>
> u：求解区间上的数值解。

比姆-沃明格式的 MATLAB 程序代码如下所示：

```
function u = peHypbBW (a,dt,n,minx,maxx,M)
%方程中的常数：a
%时间步长：dt
%空间节点个数：n
%求解区间的左端：minx
%求解区间的右端：maxx
%时间步的个数：M
%求解区间上的数值解：u
format long;
h = (maxx-minx)/(n-1);
for j=1:(n+2*M)
        u0(j) = IniU(minx+(j-2*M-1)*h);    %向左延拓 2M 个节点
end
u1 = u0;
for k=1:M
    for i=2*k+1:n+2*M
        u1(i) = u0(i)-dt*a*(u0(i)-u0(i-1))/h-a*dt*(1-a*dt/h)* ...
            (u0(i)-2*u0(i-1)+u0(i-2))/2/h;
    end
    u0 = u1;
end
u = u1((2*M+1):(2*M+n));
format short;
```

例 16-6 比姆-沃明格式求解一维对流方程应用实例。用比姆-沃明格式求解下面的初值问题：

$$\begin{cases} \dfrac{\partial u}{\partial t}+\dfrac{\partial u}{\partial x}=0 & x\in(-\infty,\infty),t>0 \\ u(x,0)=U(x) & x\in(-\infty,\infty) \end{cases}$$

其中时间步长取 0.005，求解区间为[0,1]，空间步长取 0.01，求出当 t=0.5（即 100 个时间步）时的 u 随 x 的分布图。

$$U(x)=\begin{cases} 10x+1 & -0.1\leqslant x\leqslant 0 \\ -10x+1 & 0\leqslant x\leqslant 0.1 \\ 0 & 其余 \end{cases}$$

解：先建立一个名为 IniU.m 的 MATLAB 文件，输入如下内容：

```
function ux = IniU(x)
format long;
if x<=0
    if x >= -0.1
        ux = 10*(x+0.1);
    else
        ux = 0;
    end
else
    if x <= 0.1
        ux = -10*(x-0.1);
    else
        ux = 0;
    end
end
```

然后在 MATLAB 窗口输入下列命令：

```
>> u = peHypbBW (1,0.005,101,0,1,100);
```

用比姆-沃明格式求的结果如图 16-9 所示：

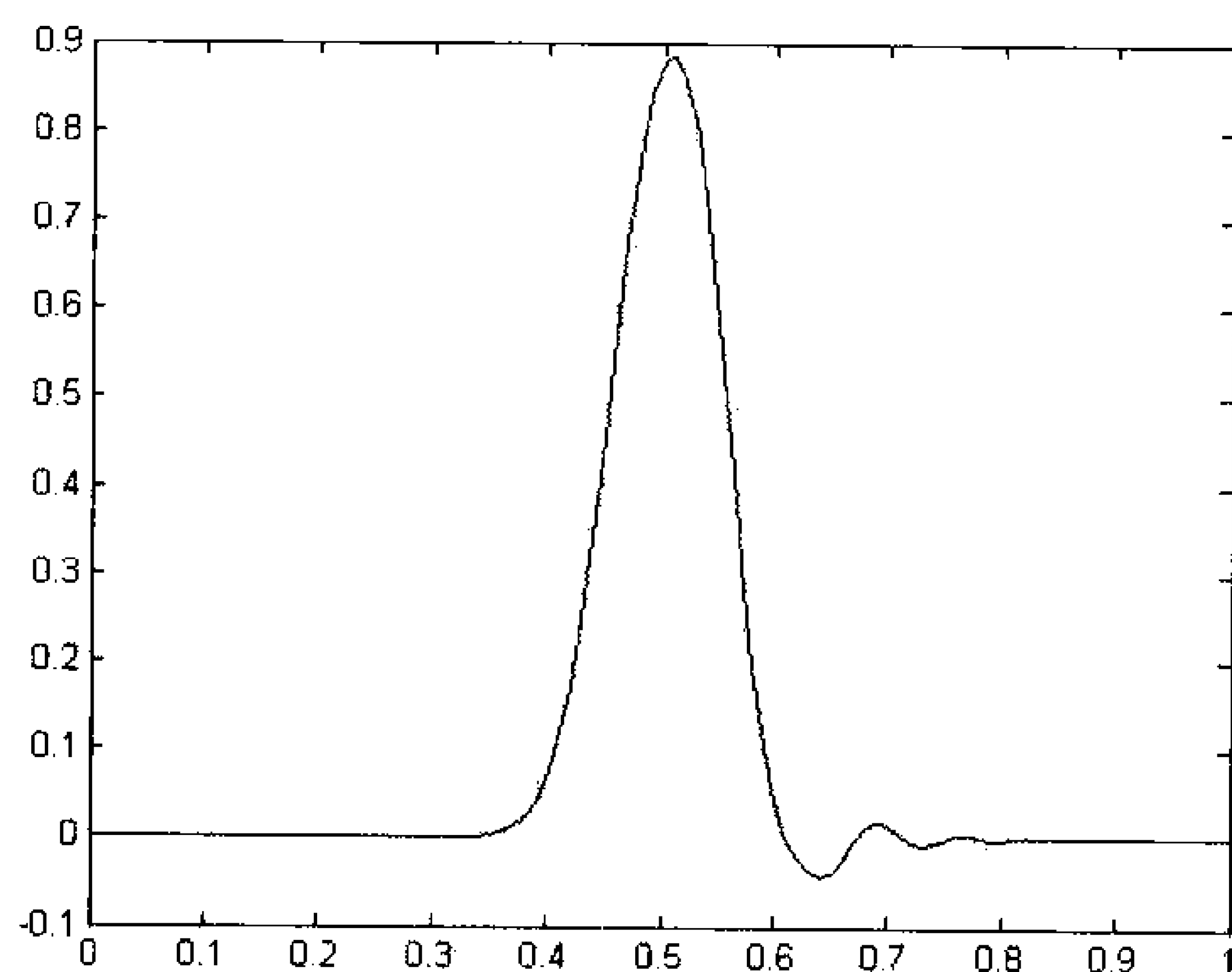


图 16-9 用比姆-沃明格式求得的当 t=0.5 时的 u 值

从图 16-9 中可以观察到，与拉克斯-温德洛夫格式相反，比姆-沃明格式的特点是函数值在右边有波动。

16.2.1.5 Richtmyer 多步格式

Richtmyer 多步格式的形式如下所示：

$$\begin{cases} \dfrac{u_j^{n+\frac{1}{2}} - \dfrac{1}{2}(u_{j+1}^n + u_{j-1}^n)}{\dfrac{\tau}{2}} + \dfrac{a}{2h}(u_{j+1}^n - u_{j-1}^n) = 0 \\ \dfrac{u_j^{n+1} - u_j^n}{\tau} + \dfrac{a}{2h}(u_{j+1}^{n+\frac{1}{2}} - u_{j-1}^{n+\frac{1}{2}}) = 0 \end{cases}$$

其中 τ 为时间步长，h 为空间步长。

Richtmyer 多步格式是需要向左和向右各延拓 2M 个节点，

在 MATLAB 中编程实现的 Richtmyer 多步格式的函数为：peHypbRich

功能：用 Richtmyer 多步格式解对流方程

调用格式：u = peHypbRich (a,dt,n,minx,maxx,M)

其中，a：方程中的常数；
- dt：时间步长；
- n：空间节点个数；
- minx：求解区间的左端；
- maxx：求解区间的右端；
- M：时间步的个数；
- u：求解区间上的数值解。

Richtmyer 多步格式的 MATLAB 程序代码如下所示：

```
function u = peHypbRich (a,dt,n,minx,maxx,M)
%方程中的常数：a
%时间步长：dt
%空间节点个数：n
%求解区间的左端：minx
%求解区间的右端：maxx
%时间步的个数：M
%求解区间上的数值解：u
format long;
h = (maxx-minx)/(n-1);
for j=1:(n+4*M)
        u0(j) = IniU(minx+(j-2*M-1)*h);     %左右各延拓 2M 个节点
end
u1 = u0;
for k=1:M
    for i=2*k+1:n+4*M-2*k
```

```
        tmpU1 = -dt*a*(u0(i+2)-u0(i))/h/4+(u0(i+2)+u0(i))/2;
        tmpU2 = -dt*a*(u0(i)-u0(i-2))/h/4+(u0(i)+u0(i-2))/2;
        u1(i) = -dt*a*(tmpU1-tmpU2)/h/2+u0(i);
    end
    u0 = u1;
end
u = u1((2*M+1):(2*M+n));
format short;
```

例 16-7 Richtmyer 多步格式求解一维对流方程应用实例。用 Richtmyer 多步格式求解下面的初值问题:

$$\begin{cases} \dfrac{\partial u}{\partial t}+\dfrac{\partial u}{\partial x}=0 & x\in(-\infty,\infty),t>0 \\ u(x,0)=U(x) & x\in(-\infty,\infty) \end{cases}$$

其中时间步长取 0.005，求解区间为[0,1]，空间步长取 0.01，求出当 t=0.5（即 100 个时间步）时的 u 随 x 的分布图。

$$U(x)=\begin{cases} 10x+1 & -0.1\leqslant x\leqslant 0 \\ -10x+1 & 0\leqslant x\leqslant 0.1 \\ 0 & \text{其余} \end{cases}$$

解：先建立一个名为 IniU.m 的 MATLAB 文件，输入如下内容：

```
function ux = IniU(x)
format long;
if x<=0
    if x >= -0.1
        ux = 10*(x+0.1);
    else
        ux = 0;
    end
else
    if x <= 0.1
        ux = -10*(x-0.1);
    else
        ux = 0;
    end
end
```

然后在 MATLAB 窗口输入下列命令：

```
>> u = peHypbRich (1,0.005,101,0,1,100);
```

用 Richtmyer 多步格式求的结果如图 16-10 所示：

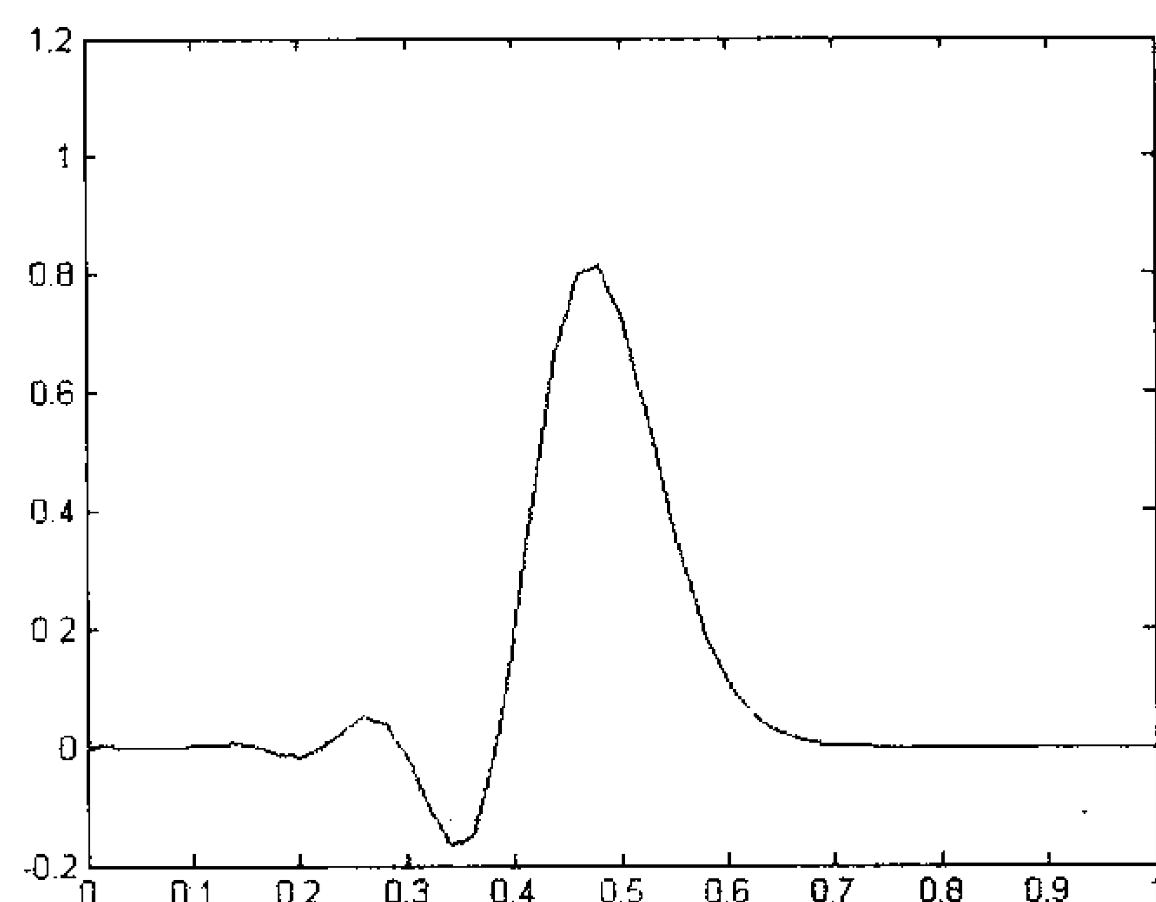


图 16-10　用 Richtmyer 多步格式求得的当 t=0.5 时的 u 值

用 Richtmyer 多步格式算出的结果并不理想，不但左边有波动，而且光滑性也不好。

16.2.1.6　拉克斯-温德洛夫多步格式

拉克斯-温德洛夫多步格式的算法公式为：

$$\begin{cases}\dfrac{u_{j+\frac{1}{2}}^{n+\frac{1}{2}}-\dfrac{1}{2}(u_{j+1}^{n}+u_{j}^{n})}{\tau}+\dfrac{a}{2h}(u_{j+1}^{n}-u_{j}^{n})=0\\[2ex] \dfrac{u_{j}^{n+1}-u_{j}^{n}}{\tau}+\dfrac{a}{h}(u_{j+\frac{1}{2}}^{n+\frac{1}{2}}-u_{j-\frac{1}{2}}^{n+\frac{1}{2}})=0\end{cases}$$

其中 τ 为时间步长，h 为空间步长。

在 MATLAB 中编程实现的拉克斯-温德洛夫多步格式的函数为：peHypbMLW

功能：用拉克斯-温德洛夫多步格式解对流方程

调用格式：u = peHypbMLW (a,dt,n,minx,maxx,M)

其中，a：方程中的常数；

dt：时间步长；

n：空间节点个数；

minx：求解区间的左端；

maxx：求解区间的右端；

M：时间步的个数；

u：求解区间上的数值解。

拉克斯-温德洛夫多步格式的 MATLAB 程序代码如下所示：

```
function u = peHypbMLW (a,dt,n,minx,maxx,M)
%方程中的常数：a
%时间步长：dt
%空间节点个数：n
%求解区间的左端：minx
%求解区间的右端：maxx
%时间步的个数：M
```

```
%求解区间上的数值解: u
format long;
h = (maxx-minx)/(n-1);
for j=1:(n+2*M)
        u0(j) = IniU(minx+(j-M-1)*h);
end
u1 = u0;
for k=1:M
    for i=k+1:n+2*M-k
        tmpU1 = -dt*a*(u0(i+1)-u0(i))/h/2+(u0(i+1)+u0(i))/2;
        tmpU2 = -dt*a*(u0(i)-u0(i-1))/h/2+(u0(i)+u0(i-1))/2;
        u1(i) = -dt*a*(tmpU1-tmpU2)/h+u0(i);
    end
    u0 = u1;
end
u = u1((M+1):(M+n));
format short;
```

例 16-8 拉克斯-温德洛夫多步格式求解一维对流方程应用实例。用拉克斯-温德洛夫多步格式求解下面的初值问题：

$$\begin{cases} \dfrac{\partial u}{\partial t}+\dfrac{\partial u}{\partial x}=0 & x\in(-\infty,\infty),t>0 \\ u(x,0)=U(x) & x\in(-\infty,\infty) \end{cases}$$

其中时间步长取 0.005，求解区间为[0,1]，空间步长取 0.01，求出当 t=0.5（即 100 个时间步）时的 u 随 x 的分布图。

$$U(x)=\begin{cases} 10x+1 & -0.1\leqslant x\leqslant 0 \\ -10x+1 & 0\leqslant x\leqslant 0.1 \\ 0 & 其余 \end{cases}$$

解：先建立一个名为 IniU.m 的 MATLAB 文件，输入如下内容：

```
function ux = IniU(x)
format long;
if x<=0
    if x >= -0.1
        ux = 10*(x+0.1);
    else
        ux = 0;
    end
else
    if x <= 0.1
        ux = -10*(x-0.1);
    else
        ux = 0;
    end
end
```

然后在 MATLAB 窗口输入下列命令：

```
>> u = peHypbMLW (1,0.005,101,0,1,100);
```

用拉克斯-温德洛夫多步格式求的结果如图 16-11 所示：

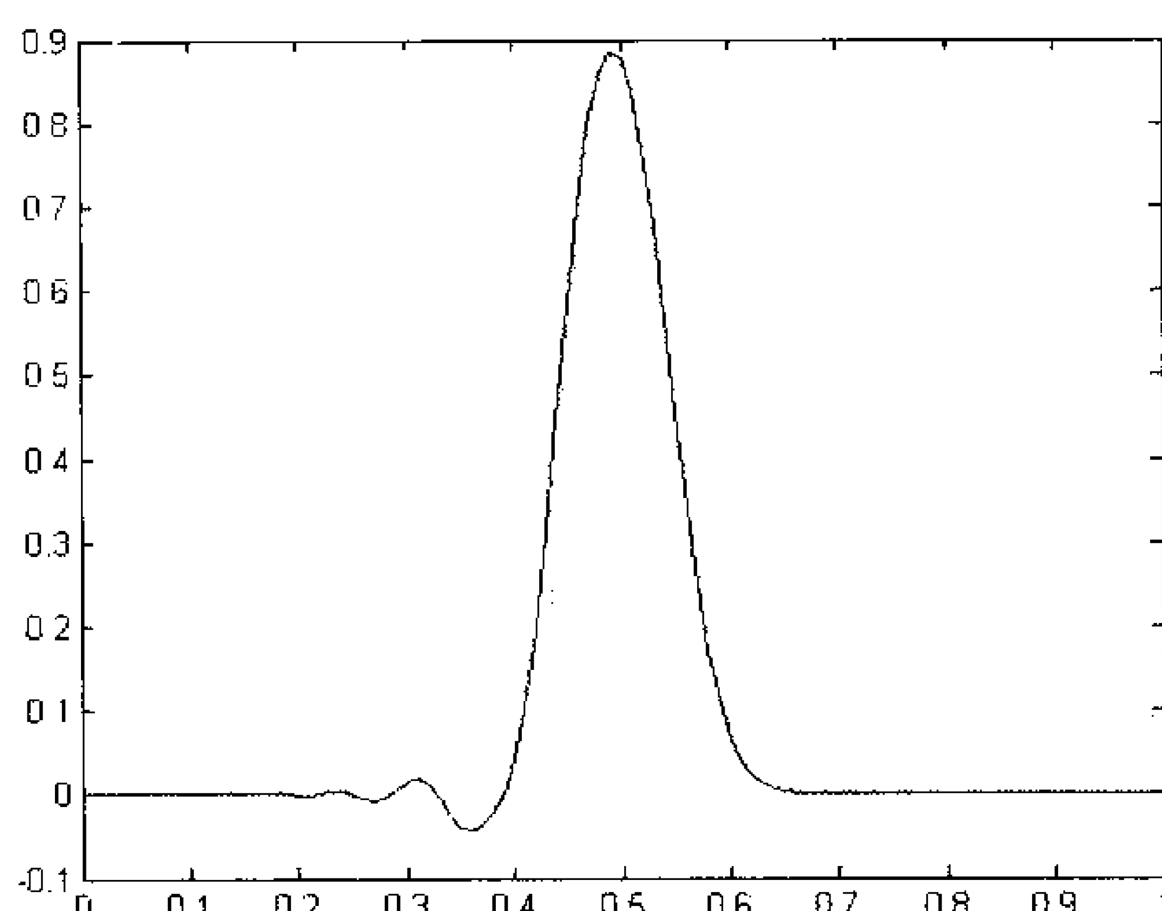


图 16-11　用拉克斯-温德洛夫多步格式求得的当 t=0.5 时的 u 值

用拉克斯-温德洛夫多步格式算出的结果比较不错，虽然左边有点小波动，但是初始函数的高度和宽度都保持得不错。

16.2.1.7　MacCormack 多步格式

MacCormack 多步格式的算法公式为：

$$\begin{cases} \dfrac{u_j^{n+\frac{1}{2}} - u_j^n}{\tau} + \dfrac{a}{h}(u_{j+1}^n - u_j^n) = 0 \\ \dfrac{u_j^{n+1} - \dfrac{1}{2}(u_j^n + u_j^{n+\frac{1}{2}})}{\tau} + \dfrac{a}{2h}(u_j^{n+\frac{1}{2}} - u_{j-1}^{n+\frac{1}{2}}) = 0 \end{cases}$$

其中 τ 为时间步长，h 为空间步长。

在 MATLAB 中编程实现的 MacCormack 多步格式的函数为：peHypbMC

功能：用 MacCormack 多步格式解对流方程

调用格式：u = peHypbMC (a,dt,n,minx,maxx,M)

其中，a：方程中的常数；

dt：时间步长；

n：空间节点个数；

minx：求解区间的左端；

maxx：求解区间的右端；

M：时间步的个数；

u：求解区间上的数值解。

MacCormack 多步格式的 MATLAB 程序代码如下所示：

```
function u = peHypbMC (a,dt,n,minx,maxx,M)
%方程中的常数：a
%时间步长：dt
```

```
%空间节点个数：n
%求解区间的左端：minx
%求解区间的右端：maxx
%时间步的个数：M
%求解区间上的数值解：u
format long;
h = (maxx-minx)/(n-1);
for j=1:(n+2*M)
    u0(j) = IniU(minx+(j-M-1)*h);
end
u1 = u0;
for k=1:M
    for i=k+1:n+2*M-k
        tmpU1 = -dt*a*(u0(i+1)-u0(i))/h+u0(i);
        tmpU2 = -dt*a*(u0(i)-u0(i-1))/h+u0(i-1);
        u1(i) = -dt*a*(tmpU1-tmpU2)/h/2+(u0(i)+tmpU1)/2;
    end
    u0 = u1;
end
u = u1((M+1):(M+n));
format short;
```

例 16-9 MacCormack 多步格式求解一维对流方程应用实例。用 MacCormack 多步格式求解下面的初值问题：

$$\begin{cases}\dfrac{\partial u}{\partial t}+\dfrac{\partial u}{\partial x}=0 & x\in(-\infty,\infty),t>0\\ u(x,0)=U(x) & x\in(-\infty,\infty)\end{cases}$$

其中时间步长取 0.005，求解区间为[0,1]，空间步长取 0.01，求出当 t=0.5（即 100 个时间步）时的 u 随 x 的分布图。

$$U(x)=\begin{cases}10x+1 & -0.1\leqslant x\leqslant 0\\ -10x+1 & 0\leqslant x\leqslant 0.1\\ 0 & 其余\end{cases}$$

解：先建立一个名为 IniU.m 的 MATLAB 文件，输入如下内容：

```
function ux = IniU(x)
format long;
if x<=0
    if x >= -0.1
        ux = 10*(x+0.1);
    else
        ux = 0;
    end
else
    if x <= 0.1
        ux = -10*(x-0.1);
    else
        ux = 0;
```

```
    end
end
```

然后在 MATLAB 窗口输入下列命令：

```
>> u = peHypbMC (1,0.005,101,0,1,100);
```

用 MacCormack 多步格式求得的结果如图 16-12 所示：

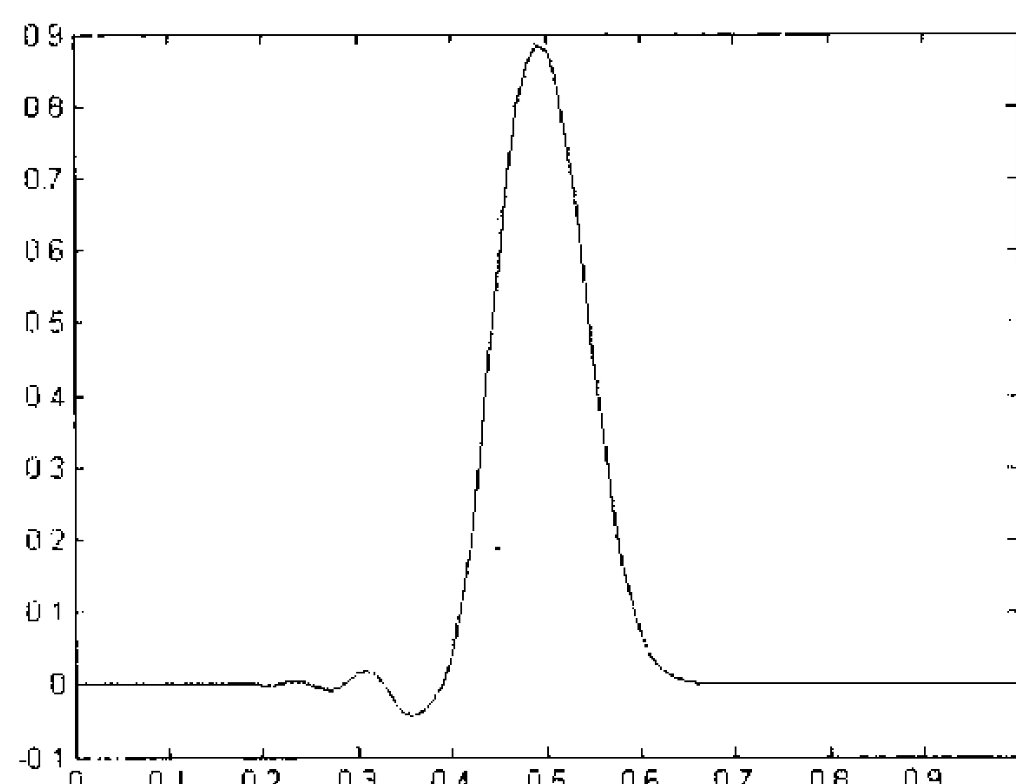


图 16-12 用 MacCormack 多步格式求得的当 t=0.5 时的 u 值

MacCormack 多步格式的结果与拉克斯-温德洛夫多步格式求得的结果差不多。

16.2.2 二维对流方程

二维对流方程的形式如下所示：

$$\frac{\partial u}{\partial t}+a\frac{\partial u}{\partial x}+b\frac{\partial u}{\partial y}=0 \quad -\infty<x,y<+\infty,t>0\ (\ a,b\ \text{为常数}\ )$$

如果给定初始条件：

$$u(x,y,0)=U(x,y) \quad -\infty<x,y<+\infty$$

则二维对流方程的通解为：

$$u(x,y,t)=U(x-at,y-bt) \quad -\infty<x,y<+\infty,t>0$$

下面介绍两种常见的求解二维对流方程的拉克斯-弗里德里希斯格式和近似分裂格式。

16.2.2.1 拉克斯-弗里德里希斯格式

拉克斯-弗里德里希斯格式的形式如下所示：

$$\frac{u_{j,k}^{n+1}-\frac{1}{4}(u_{j+1,k}^{n}+u_{j-1,k}^{n}+u_{j,k+1}^{n}+u_{j,k-1}^{n})}{\tau}+$$

$$\frac{a}{2h_x}(u_{j+1,k}^{n}-u_{j-1,k}^{n})+\frac{b}{2h_y}(u_{j,k+1}^{n}-u_{j,k-1}^{n})=0$$

其中 τ 为时间步长，h 为空间步长。

同一维对流方程一样，二维的拉克斯-弗里德里希斯格式也需要进行节点的延拓，延拓的方法可参看程序。

在 MATLAB 中编程实现的二维对流方程拉克斯-弗里德里希斯格式的函数为：peHypb2LF

功能：用拉克斯-弗里德里希斯格式解二维对流方程的初值问题

调用格式：u = peHypb2LF (a,b,dt,nx,minx,maxx,ny,miny,maxy,M)

其中，a：方程中的常数 1；

b：方程中的常数 2；

dt：时间步长；

nx：x 方向节点个数；

minx：x 求解区间的左端；

maxx：x 求解区间的右端；

ny：y 方向节点个数；

miny：y 求解区间的左端；

maxy：y 求解区间的右端；

M：时间步个数；

u：求解区间上的数值解。

拉克斯-弗里德里希斯格式求解二维对流方程初值问题的 MATLAB 程序代码如下所示：

```
function u = peHypb2LF (a,b,dt,nx,minx,maxx,ny,miny,maxy,M)
%方程中的常数 1：a
%方程中的常数 2：b
%时间步长：dt
%x 方向节点个数：nx
%x 求解区间的左端：minx
%x 求解区间的右端：maxx
%y 方向节点个数：nx
%y 求解区间的左端：miny
%y 求解区间的右端：maxy
%时间步的个数：M
%求解区间上的数值解：u
format long;
hx = (maxx-minx)/(nx-1);
hy = (maxy-miny)/(ny-1);
for i=1:nx+2*M
    for j=1:(ny+2*M)
        u0(i,j) = Ini2U(minx+(i-M-1)*hx,miny+(j-M-1)*hy);    %二维节点延拓
    end
end
u1 = u0;
for k=1:M
    for i=k+1:nx+2*M-k
        for j=k+1:ny+2*M-k
            u1(i,j) = (u0(i+1,j)+u0(i-1,j)+u0(i,j+1)+u0(i,j-1))/4 ...
```

```
                -a*dt*(u0(i+1,j)-u0(i-1,j))/2/hx ...
                -b*dt*(u0(i,j+1)-u0(i,j-1))/2/hy;
        end
    end
    u0 = u1;
end
u = u1((M+1):(M+nx),(M+1):(M+ny));
format short;
```

例 16-10 拉克斯-弗里德里希斯格式求解二维对流方程应用实例。用拉克斯-弗里德里希斯格式求解下面二维对流方程的初值问题：

$$\begin{cases} \dfrac{\partial u}{\partial t}+\dfrac{\partial u}{\partial x}+\dfrac{\partial u}{\partial y}=0 & -\infty < x, y < +\infty, t > 0 \\ u(x, y, 0)=e^{-(10x^2+10y^2)} & x, y \in (-\infty, \infty) \end{cases}$$

其中时间步长取 0.005，求解区间为[0,1]，空间步长取 0.01。求出当 t=0.5（即 100 个时间步）时的 u 随 x, y 的分布图。

解：先建立一个名为 Ini2U.m 的 MATLAB 文件，输入如下内容：

```
function uxy = Ini2U(x,y)
format long;
uxy = exp(-10*x*x-10*y*y);
```

然后在 MATLAB 窗口输入下列命令：

```
>> u = peHypb2LF (1,1,0.005,101,0,1, 101,0,1,100);
```

初始值（即 t =0 时）和用拉克斯-弗里德里希斯格式求的结果如图 16-13 及图 16-14 所示：

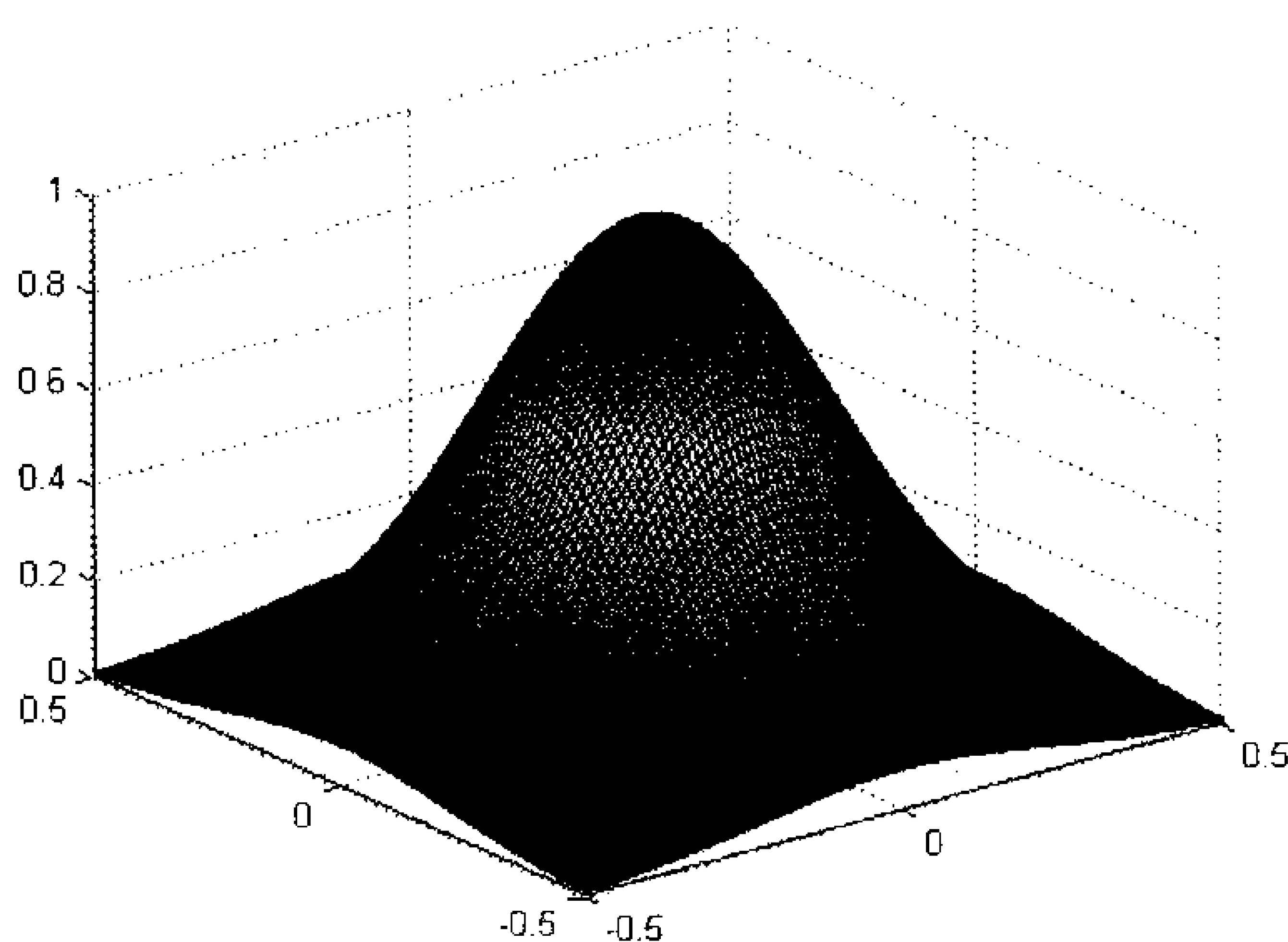


图 16-13　t =0 时的 u 值

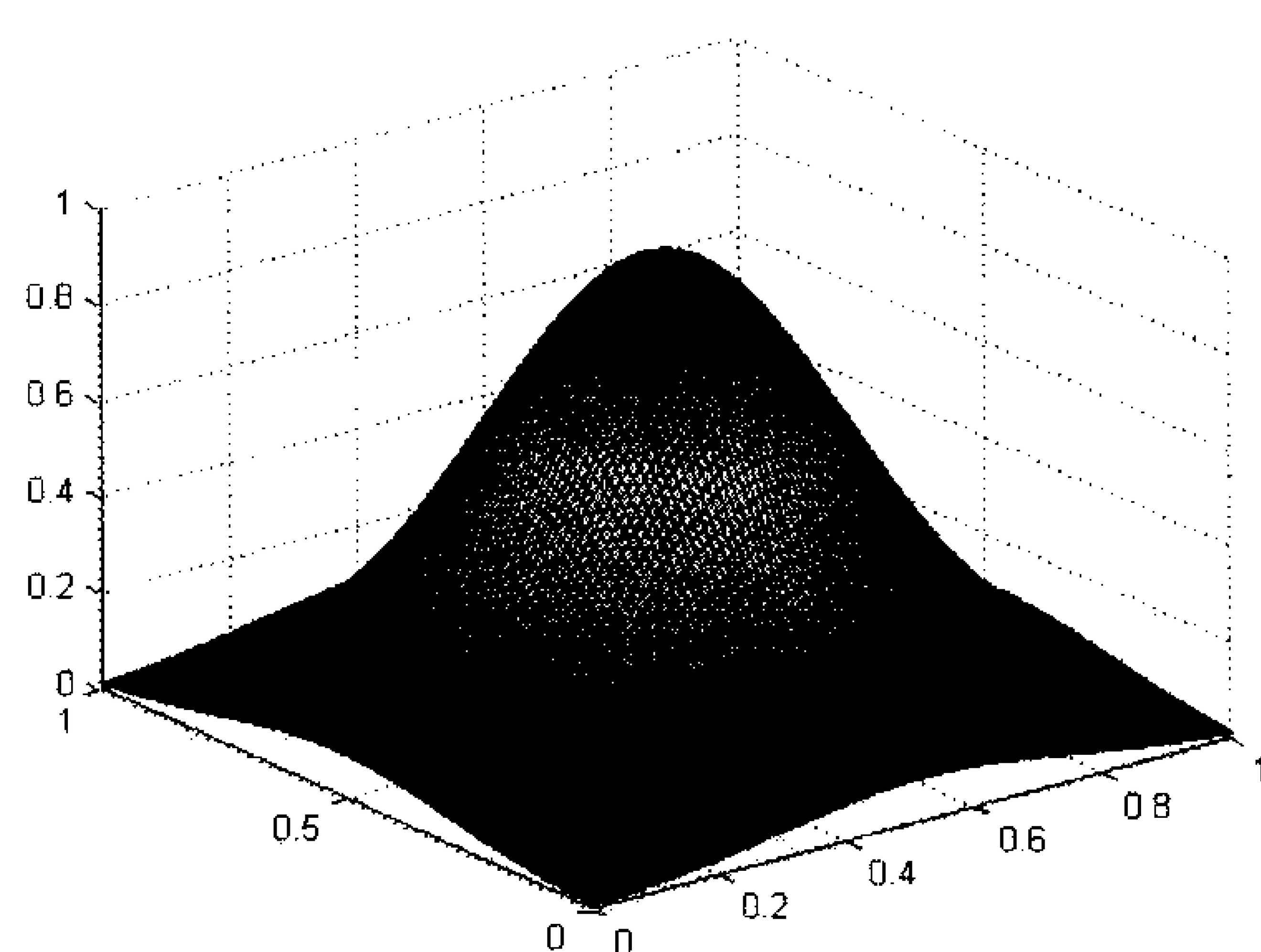


图 16-14　t =0.5 时的 u 值

从结果的对比可以看出，拉克斯-弗里德里希斯格式算出的结果非常好。

16.2.2.2 近似分裂格式

近似分裂格式的形式如下所示：

$$\begin{cases} u_{j,k}^{n+\frac{1}{2}} = u_{j,k}^{n} - \dfrac{a\tau}{2h_x}(u_{j+1,k}^{n} - u_{j-1,k}^{n}) + \dfrac{a^2\tau^2}{2{h_x}^2}(u_{j+1,k}^{n} - 2u_{j,k}^{n} + u_{j-1,k}^{n}) \\ u_{j,k}^{n+1} = u_{j,k}^{n+\frac{1}{2}} - \dfrac{b\tau}{2h_y}(u_{j,k+1}^{n+\frac{1}{2}} - u_{j,k-1}^{n+\frac{1}{2}}) + \dfrac{b^2\tau^2}{2{h_y}^2}(u_{j,k+1}^{n+\frac{1}{2}} - 2u_{j,k}^{n+\frac{1}{2}} + u_{j,k-1}^{n+\frac{1}{2}}) \end{cases}$$

其中 τ 为时间步长，h 为空间步长。

在 MATLAB 中编程实现的二维对流方程拉克斯-弗里德里希斯格式的函数为：peHypb2FL

功能：用拉克斯-弗里德里希斯格式解二维对流方程的初值问题

调用格式：u = peHypb2FL (a,b,dt,nx,minx,maxx,ny,miny,maxy,M)

其中，a：方程中的常数 1；
- b：方程中的常数 2；
- dt：时间步长；
- nx：x 方向节点个数；
- minx：x 求解区间的左端；
- maxx：x 求解区间的右端；
- ny：y 方向节点个数；
- miny：y 求解区间的左端；
- maxy：y 求解区间的右端；
- M：时间步个数；
- u：求解区间上的数值解。

拉克斯-弗里德里希斯格式求解二维对流方程初值问题的 MATLAB 程序代码如下所示：

```
function u = peHypb2FL (a,b,dt,nx,minx,maxx,ny,miny,maxy,M)
%方程中的常数 1：a
%方程中的常数 2：b
%时间步长：dt
%x 方向节点个数：nx
%x 求解区间的左端：minx
%x 求解区间的右端：maxx
%y 方向节点个数：nx
%y 求解区间的左端：miny
%y 求解区间的右端：maxy
%时间步的个数：M
%求解区间上的数值解：u
format long;
hx = (maxx-minx)/(nx-1);
hy = (maxy-miny)/(ny-1);
for i=1:nx+4*M
    for j=1:(ny+4*M)
        u0(i,j) = Ini2U(minx+(i-2*M-1)*hx,miny+(j-2*M-1)*hy);   %节点的延拓
```

```
        end
    end
    u1 = u0;
    for k=1:M
        for i=2*k+1:nx+4*M-2*k
            for j=2*k-1:ny+4*M-2*k+2
                tmpU(i,j) = u0(i,j) - a*dt*(u0(i+1,j)-u0(i-1,j))/2/hx + ...
                    (a*dt/hx)^2*(u0(i+1,j)-2*u0(i,j)+u0(i-1,j))/2;
            end
        end
        for i=2*k+1:nx+4*M-2*k
            for j=2*k+1:nx+4*M-2*k
                u1(i,j) = tmpU(i,j) - b*dt*(tmpU(i,j+1)-tmpU(i,j-1))/2/hy + ...
                    (b*dt/hy)^2*(tmpU(i,j+1)-2*tmpU(i,j)+tmpU(i,j-1))/2;
            end
        end
        u0 = u1;
    end
    u = u1((2*M+1):(2*M+nx),(2*M+1):(2*M+ny));
    format short;
```

例 16-11　近似分裂格式求解二维对流方程应用实例。用近似分裂格式求解下面的初值问题：

$$\begin{cases}\dfrac{\partial u}{\partial t}+\dfrac{\partial u}{\partial x}+\dfrac{\partial u}{\partial y}=0 & -\infty<x,y<+\infty,t>0\\ u(x,y,0)=e^{-(10x^2+10y^2)} & x,y\in(-\infty,\infty)\end{cases}$$

其中时间步长取 0.005，求解区间为[0,1]，空间步长取 0.01。求出当 t=0.5（即 100 个时间步）时的 u 随 x, y 的分布图。

解：先建立一个名为 Ini2U.m 的 MATLAB 文件，输入如下内容：

```
function uxy = Ini2U(x,y)
format long;
uxy = exp(-10*x*x-10*y*y);
```

然后在 MATLAB 窗口输入下列命令：

```
>> u = peHypb2FL (1,1,0.005,101,0,1, 101,0,1,100);
```

用近似分裂格式求的结果如图 16-15 所示：

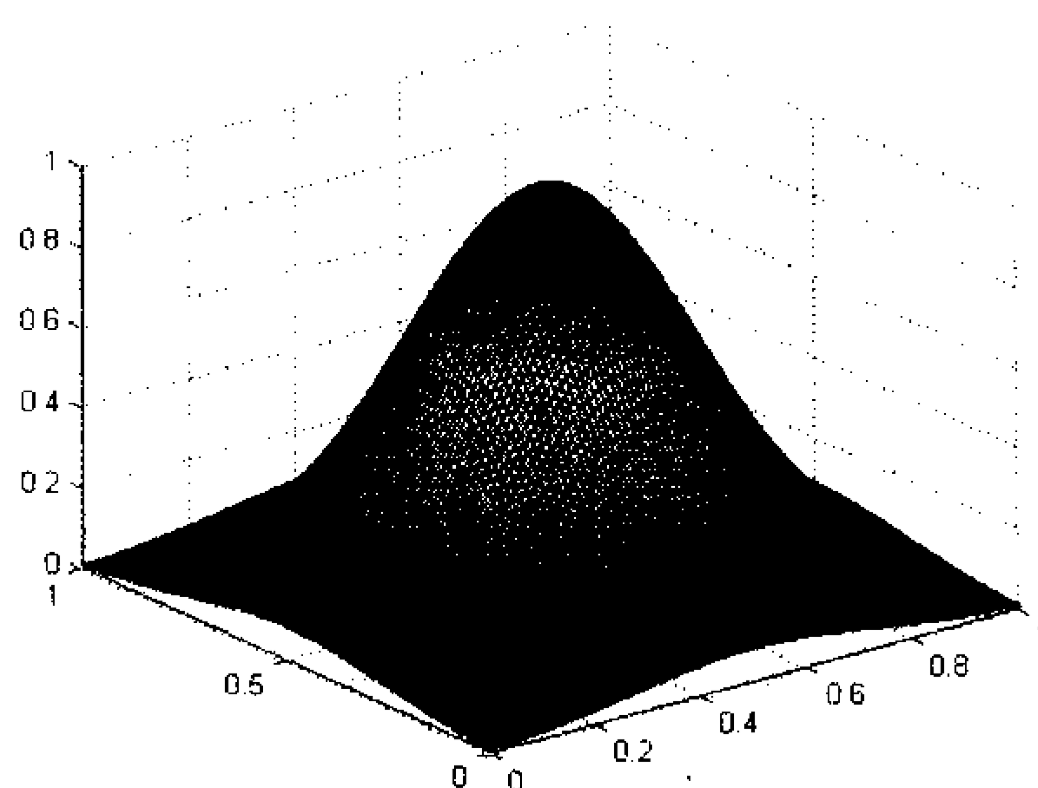


图 16-15　用近似分裂格式求得的在求解域上的当 t=0.5 时的场函数数值图

近似分裂格式也是一种不错的格式，其结果也非常接近理论值。

16.3 抛物线偏微分方程

在实际应用中遇到的抛物线偏微分方程（后面也有简称为抛物方程）主要是扩散方程，扩散方程有很强的物理背景，例如不同浓度物质之间的扩散过程、热传递过程、波传播等过程都可用扩散过程来描述。因此，本节以扩散方程和对流扩散方程为例介绍扩散方程的几种常用差分格式。

16.3.1 扩散方程

扩散方程是最简单的抛物线偏微分方程，其形式如下所示：

$$\frac{\partial u}{\partial t}=c\frac{\partial^2 u}{\partial x^2}\quad x\in(-\infty,\infty),t\geqslant 0,c\text{ 为常数}$$

如果给定初始条件：

$$u(x,0)=U(x)\quad x\in(-\infty,\infty)$$

则扩散方程的通解为：

$$u(x,t)=\frac{1}{\sqrt{4\pi ct}}\int_{-\infty}^{+\infty}e^{-\frac{(x-\xi)^2}{4ct}}U(\xi)d\xi\quad x\in(-\infty,\infty),t>0$$

在抛物线偏微分方程的求解差分格式中，隐式格式比较居多，因此下面的格式中比较重要的是求解扩散方程边值问题的几种隐式格式。

16.3.1.1 显式格式

显式格式的形式如下所示：

$$\frac{u_j^{n+1}-u_j^n}{\tau}-\frac{c}{h^2}(u_{j+1}^n-2u_j^n+u_{j-1}^n)=0$$

其中 τ 为时间步长，h 为空间步长。

显示格式很简单，但是其精度很差，而且求得的解容易出现振荡，这可以从例题中看出来。

显式格式用来求如下形式的抛物问题：

$$\begin{cases}\dfrac{\partial u}{\partial t}=c\dfrac{\partial^2 u}{\partial x^2} & x\in(-\infty,\infty),t\geqslant 0\\ u(x,0)=U(x) & x\in(-\infty,\infty)\end{cases}$$

在 MATLAB 中编程实现的扩散方程显式格式的函数为：peParabExp

功能：用显式格式解扩散方程的初值问题

调用格式：u = peParabExp(c,dt,n,minx,maxx,M)

其中，c：方程中的常数；

dt：时间步长；

n：空间节点的个数；
minx：求解区间的左端；
maxx：求解区间的右端；
M：时间步个数；
u：求解区间上的数值解。

显式格式求解扩散方程初值问题的 MATLAB 程序代码如下所示：

```
function u = peParabExp(c,dt,n,minx,maxx,M)
%方程中的常数：c
%时间步长：dt
%空间节点个数：n
%求解区间的左端：minx
%求解区间的右端：maxx
%时间步的个数：M
%求解区间上的数值解：u
format long;
h = (maxx-minx)/(n-1);
for j=1:(n+2*M)
    u0(j) = PrIniU(minx+(j-M-1)*h);        %节点延拓
end
u1 = u0;
for k=1:M
    for i=k+1:n+2*M-k
        u1(i) = dt*c*(u0(i+1)-2*u0(i)+u0(i-1))/h/h+u0(i);
    end
    u0 = u1;
end
u = u1((M+1):(M+n));
format short;
```

例 16-12 显式格式求解扩散方程应用实例。用显式格式求解下面扩散方程的初值问题：

$$\begin{cases}\dfrac{\partial u}{\partial t}=\dfrac{\partial^2 u}{\partial x^2} & -\infty<x<+\infty,t>0\\ u(x,0)=\sin x & x\in(-\infty,\infty)\end{cases}$$

其中时间步长取 0.005，求解区间为[0,1]，空间步长取 0.01。求出当 t=0.5（即 100 个时间步）时的 u 随 x 的分布图。

解：先建立一个名为 PrIniU.m 的 MATLAB 文件，输入如下内容：

```
function ux = PrIniU(x)
format long;
ux = sin(x);
```

然后在 MATLAB 窗口输入下列命令：

```
>> u = peParabExp(1,0.005,101,0,1,100);
```

用显式格式求得的结果如图 16-16 所示：

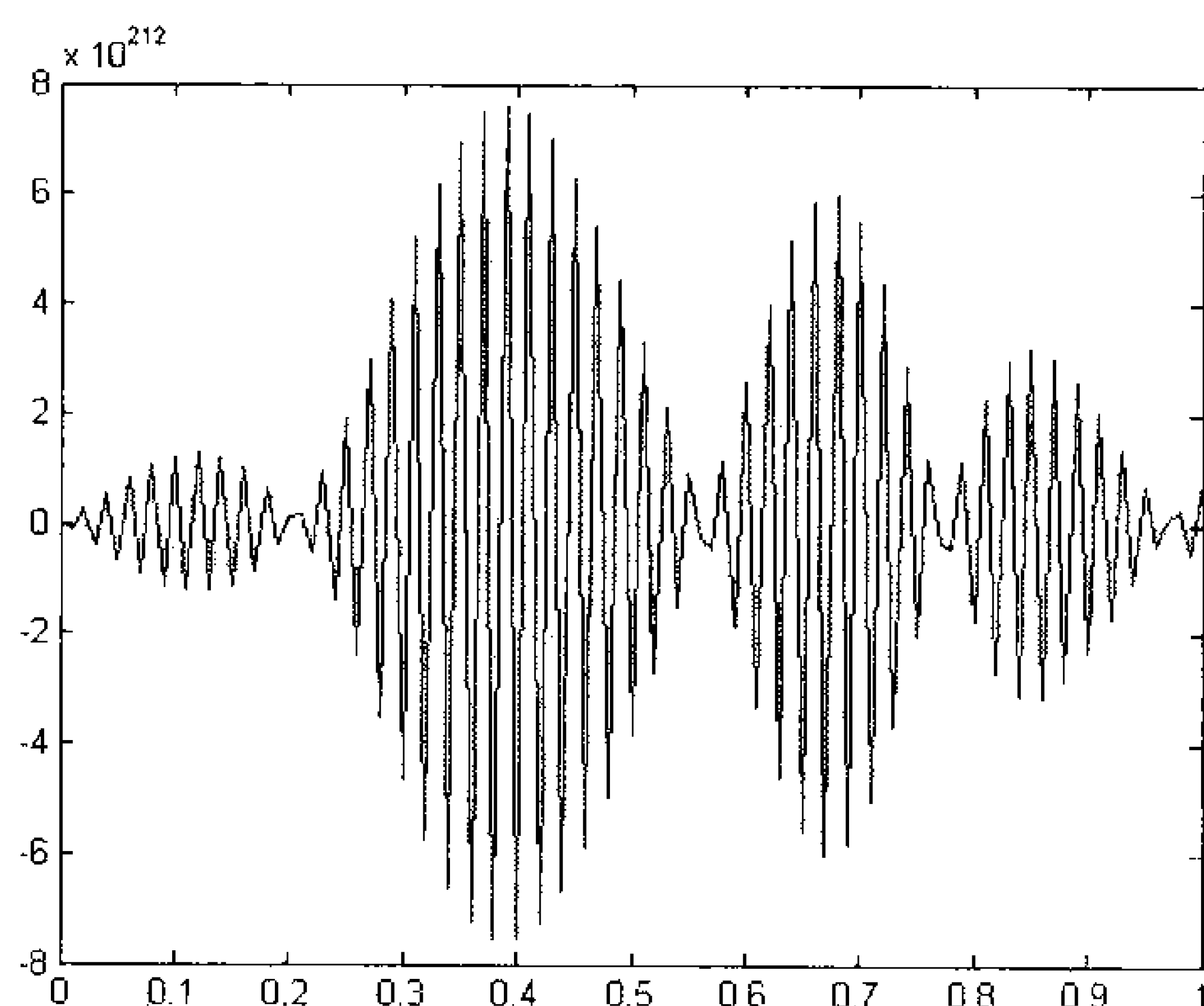


图 16-16 例 16-12 用显式格式求得的当 t=0.5 时的在求解域上的场函数值图

数值结果振荡得非常厉害，说明显式格式在这种条件下不稳定。

16.3.1.2 跳点格式

跳点格式的形式如下所示：

$$\begin{cases} \dfrac{u_j^{n+1}-u_j^n}{\tau}-\dfrac{c}{h^2}(u_{j+1}^n-2u_j^n+u_{j-1}^n)=0 \\ \qquad n+1+j=\text{偶数}, j=0,1,\cdots,n \\ \dfrac{u_j^{n+1}-u_j^n}{\tau}-\dfrac{c}{h^2}(u_{j+1}^{n+1}-2u_j^{n+1}+u_{j-1}^{n+1})=0 \\ \qquad n+1+j=\text{奇数}, j=0,1,\cdots,n \end{cases}$$

其中 τ 为时间步长，h 为空间步长。

跳点格式其实也是一种显示格式，跳点格式用来求如下形式的抛物问题：

$$\begin{cases} \dfrac{\partial u}{\partial t}=c\dfrac{\partial^2 u}{\partial x^2} & x\in(-\infty,\infty), t\geqslant 0 \\ u(x,0)=U(x) & x\in(-\infty,\infty) \end{cases}$$

在 MATLAB 中编程实现的扩散方程跳点格式的函数为：peParabTD

功能：用跳点格式解扩散方程的初值问题

调用格式：u = peParabTD(c,dt,n,minx,maxx,M)

其中，c：方程中的常数；

dt：时间步长；

n：空间节点个数；

minx：求解区间的左端；

maxx：求解区间的右端；

M：时间步的个数；

u：求解区间上的数值解。

跳点格式求解扩散方程初值问题的 MATLAB 程序代码如下所示：

```
function u = peParabTD(c,dt,n,minx,maxx,M)
%方程中的常数：c
%时间步长：dt
%空间节点个数：n
%求解区间的左端：minx
%求解区间的右端：maxx
%时间步的个数：M
%求解区间上的数值解：u
format long;
h = (maxx-minx)/(n-1);
u0(1) = lbu;
u0(n) = rbu;
for j=2:n-1
    u0(j) = PrIniU(minx+(j-1)*h);
end
u1 = u0;
for k=1:M
    for i=2:n-1
        if mod(n+i,2) == 0
            u1(i) = u0(i) + c*dt*(u0(i+1) - 2*u0(i) + u0(i-1))/h/h;
            if i > 2
               u1(i-1) =( u0(i-1) + c*dt*(u1(i) + u1(i-2))/h/h )/(1 +
2*c*dt/h/h);
            end
        end
    end
    u0 = u1;
end
u = u1((M+1):(M+n));
format short;
```

例 16-13 跳点格式求解扩散方程应用实例。用跳点格式求解下面扩散方程的初值问题：

$$\begin{cases}\dfrac{\partial u}{\partial t}=\dfrac{\partial^2 u}{\partial x^2} & -\infty<x<+\infty,t>0\\ u(x,0)=\sin x & x\in(-\infty,\infty)\end{cases}$$

其中时间步长取 0.005，求解区间为[0,1]，空间步长取 0.01。求出当 t=0.5（即 100 个时间步）时的 u 随 x 的分布图。

解：先建立一个名为 PrIniU.m 的 MATLAB 文件，输入如下内容：

```
function ux = PrIniU(x)
format long;
ux = sin(x);
```

然后在 MATLAB 窗口输入下列命令：

```
>> u = peParabTD(1,0.005,101,0,1,100);
```

用跳点格式求得的结果如图 16-17 所示：

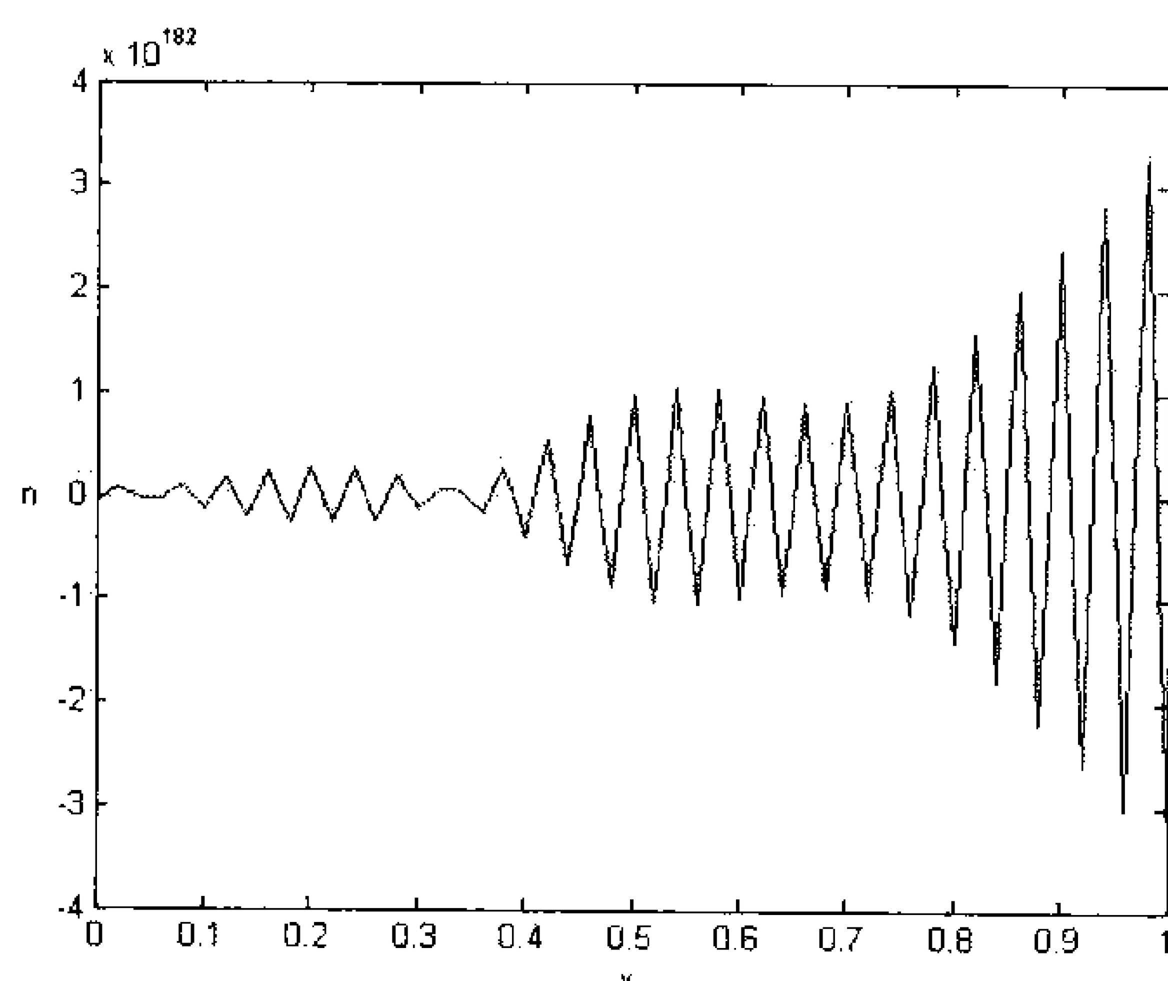


图 16-17 例 16-13 用跳点格式求得的当 t=0.5 时的在求解域上的场函数值图

数值结果振荡得非常厉害，说明跳点格式在这种条件下也不稳定。

16.3.1.3 隐式格式

隐式格式的形式如下所示：

$$\frac{u_j^{n+1}-u_j^n}{\tau}-\frac{c}{h^2}(u_{j+1}^{n+1}-2u_j^{n+1}+u_{j-1}^{n+1})=0$$

其中 τ 为时间步长，h 为空间步长。

隐式格式的每一步都要求解一个线性方程组，即它并不能显式地从当前值得到下一个时间步的函数值，但是在每一个时间步中，边界条件都不变，也就是端点处的函数值不变。

隐式格式用来求如下形式的抛物问题：

$$\begin{cases}\dfrac{\partial u}{\partial t}=c\dfrac{\partial^2 u}{\partial x^2} & x\in[x_1,x_2],t>0\\ u(x_1,t)=u_1 & t>0\\ u(x_2,t)=u_2 & t>0\\ u(x,0)=U(x) & x\in[x_1,x_2]\end{cases}$$

在 MATLAB 中编程实现的扩散方程隐式格式的函数为：peParabImp

功能：用隐式格式解扩散方程的初值问题

调用格式：u = peParabImp(c,dt,n,minx,maxx,lbu,rbu,M)

其中，c：方程中的常数；

dt：时间步长；

n：空间节点个数；

minx：求解区间的左端；

maxx：求解区间的右端；

lbu：左边界条件；

rbu：右边界条件；

M：时间步的个数；
u：求解区间上的数值解。

隐式格式求解扩散方程初值问题的 MATLAB 程序代码如下所示：

```
function u = peParabImp(c,dt,n,minx,maxx,lbu,rbu,M)
%方程中的常数：c
%时间步长：dt
%空间节点个数：n
%求解区间的左端：minx
%求解区间的右端：maxx
%左边界条件：lbu
%右边界条件：rbu
%时间步的个数：M
%求解区间上的数值解：u
format long;
h = (maxx-minx)/(n-1);
u0(1) = lbu;
u0(n) = rbu;
for j=2:n-1
    u0(j) = PrIniU(minx+(j-1)*h);
end
u1 = u0;
for k=1:M
    A = zeros(n-2,n-2);
    cb =  - transpose(u0(2:(n-1)));
    cb(1) = cb(1) - dt*c*lbu/h/h;
    cb(n-2) = cb(n-2) - dt*c*rbu/h/h;
    A(1,1) = -2*dt*c/h/h -1;
    A(1,2) = dt*c/h/h ;
    for i=2:n-3
        A(i,i-1) = dt*c/h/h ;
        A(i,i) = - 2*dt*c/h/h -1 ;
        A(i,i+1) = dt*c/h/h  ;
    end
    A(n-2,n-2) = -2*dt*c/h/h -1;
    A(n-2,n-3) = dt*c/h/h;
    u1(2:(n-1)) = A\cb;                    %通过求解方程组得到当前时间步的函数值
    u0 = u1;
end
u = u1;
format short;
```

例 16-14 隐式格式求解扩散方程应用实例。用隐式格式求解下面扩散方程的初值问题：

$$\begin{cases} \dfrac{\partial u}{\partial t}=\dfrac{\partial^2 u}{\partial x^2} & -\infty<x<+\infty, t>0 \\ u(x,0)=\sin x & x\in(0,1) \\ u(0,t)=0 \\ u(1,t)=1 \end{cases}$$

其中时间步长取 0.005，空间步长取 0.01。求出当 t=0.5（即 100 个时间步）时的 u 随 x 的分布图。

解：先建立一个名为 PrIniU.m 的 MATLAB 文件，输入如下内容：

```
function ux = PrIniU(x)
format long;
ux = sin(x);
```

然后在 MATLAB 窗口输入下列命令：

```
>> u = peParabImp(1,0.005,101,0,1,100)
```

用隐式格式求得的结果如图 16-18 所示：

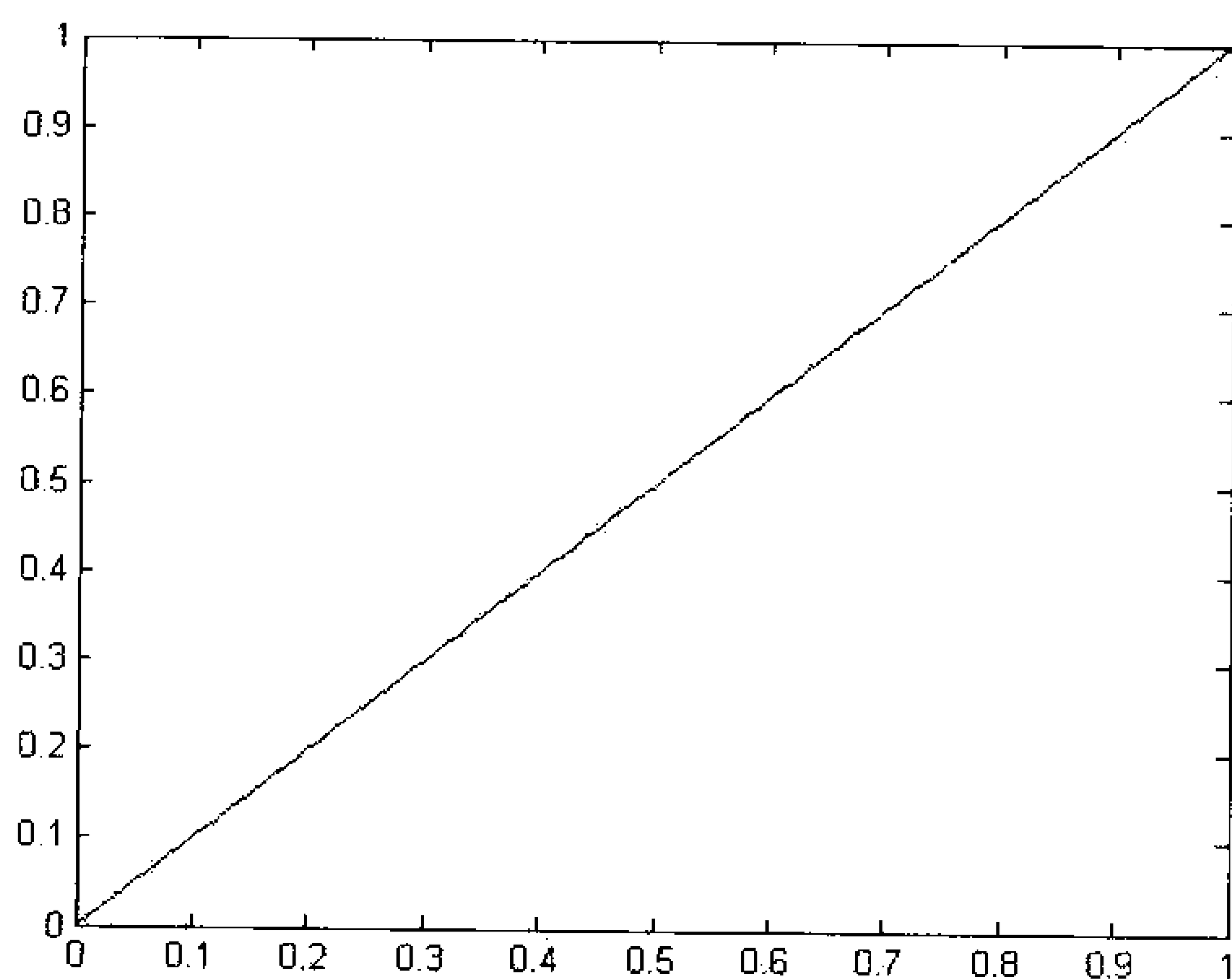


图 16-18　用隐式格式求得的当 t=0.5 时的在求解域上的场函数值图

结果是一条比较稳定的直线。

16.3.1.4　克拉克-尼科尔森格式

克拉克-尼科尔森格式的形式如下所示：

$$\frac{u_j^{n+1}-u_j^n}{\tau}-\frac{c}{h^2}(u_{j+1}^{n+1}-2u_j^{n+1}+u_{j-1}^{n+1})=0$$

其中 τ 为时间步长，h 为空间步长。

克拉克-尼科尔森格式用来求解如下形式的抛物问题：

$$\begin{cases} \dfrac{\partial u}{\partial t}=c\dfrac{\partial^2 u}{\partial x^2} & x\in[x_1,x_2], t>0 \\ u(x_1,t)=u_1 & t>0 \\ u(x_2,t)=u_2 & t>0 \\ u(x,0)=U(x) & x\in[x_1,x_2] \end{cases}$$

在 MATLAB 中编程实现的扩散方程克拉克-尼科尔森格式的函数为：peParabKN

功能：用克拉克-尼科尔森格式解扩散方程的初值问题

调用格式：u = peParabKN (c,dt,n,minx,maxx,lbu,rbu,M)

其中，c：方程中的常数；

dt：时间步长；

n：空间节点个数；

minx：求解区间的左端；

maxx：求解区间的右端；

lbu：左边界条件；

rbu：右边界条件；

M：时间步的个数；

u：求解区间上的数值解。

克拉克-尼科尔森格式求解扩散方程的初值问题的 MATLAB 程序代码如下所示：

```
function u = peParabKN(c,dt,n,minx,maxx,lbu,rbu,M)
%方程中的常数：c
%时间步长：dt
%空间节点个数：n
%求解区间的左端：minx
%求解区间的右端：maxx
%左边界条件：lbu
%右边界条件：rbu
%时间步的个数：M
%求解区间上的数值解：u
format long;
h = (maxx-minx)/(n-1);
u0(1) = lbu;
u0(n) = rbu;
for j=2:n-1
    u0(j) = PrIniU(minx+(j-1)*h);
end
u1 = u0;
for k=1:M
    A = zeros(n-2,n-2);
    cb =  zeros(n-2,1);
    cb(1) = -u0(2) -(u0(3)-2*u0(2)+u0(1))*dt*c*lbu/h/h/2 - dt*c*lbu/h/h/2;
    cb(n-2) = -u0(n-1) -(u0(n)-2*u0(n-1)+u0(n-2))*dt*c*lbu/h/h/2 -   ...
                  dt*c*rbu/h/h/2;
    for i=2:n-3
        cb(i) = -u0(i+1) -(u0(i+2)-2*u0(i+1)+u0(i))*dt*c*lbu/h/h/2;
    end
    A(1,1) = -dt*c/h/h -1;
    A(1,2) = dt*c/h/h/2 ;
    for i=2:n-3
        A(i,i-1) = dt*c/h/h/2 ;
```

```
        A(i,i) = - dt*c/h/h -1 ;
        A(i,i+1) = dt*c/h/h/2  ;
    end
    A(n-2,n-2) = -dt*c/h/h -1;
    A(n-2,n-3) = dt*c/h/h/2;
    u1(2:(n-1)) = A\cb;
    u0 = u1;
end
u = u1;
format short;
```

例 16-15 克拉克-尼科尔森格式求解扩散方程应用实例。用克拉克-尼科尔森格式求解下面扩散方程的初值问题：

$$\begin{cases} \dfrac{\partial u}{\partial t}=\dfrac{\partial^2 u}{\partial x^2} & -\infty < x < +\infty, t>0 \\ u(x,0)=\sin x & x\in(0,1) \\ u(0,t)=0 \\ u(1,t)=1 \end{cases}$$

其中时间步长取 0.005，空间步长取 0.01。求出当 t=0.5（即 100 个时间步）时的 u 随 x 的分布图。

解：先建立一个名为 PrIniU.m 的 MATLAB 文件，输入如下内容：

```
function ux = PrIniU(x)
format long;
ux = sin(x);
```

然后在 MATLAB 窗口输入下列命令：

```
>> u = peParabKN(1,0.005,101,0,1,0,1,100)
```

用克拉克-尼科尔森格式求得的结果如图 16-19 所示：

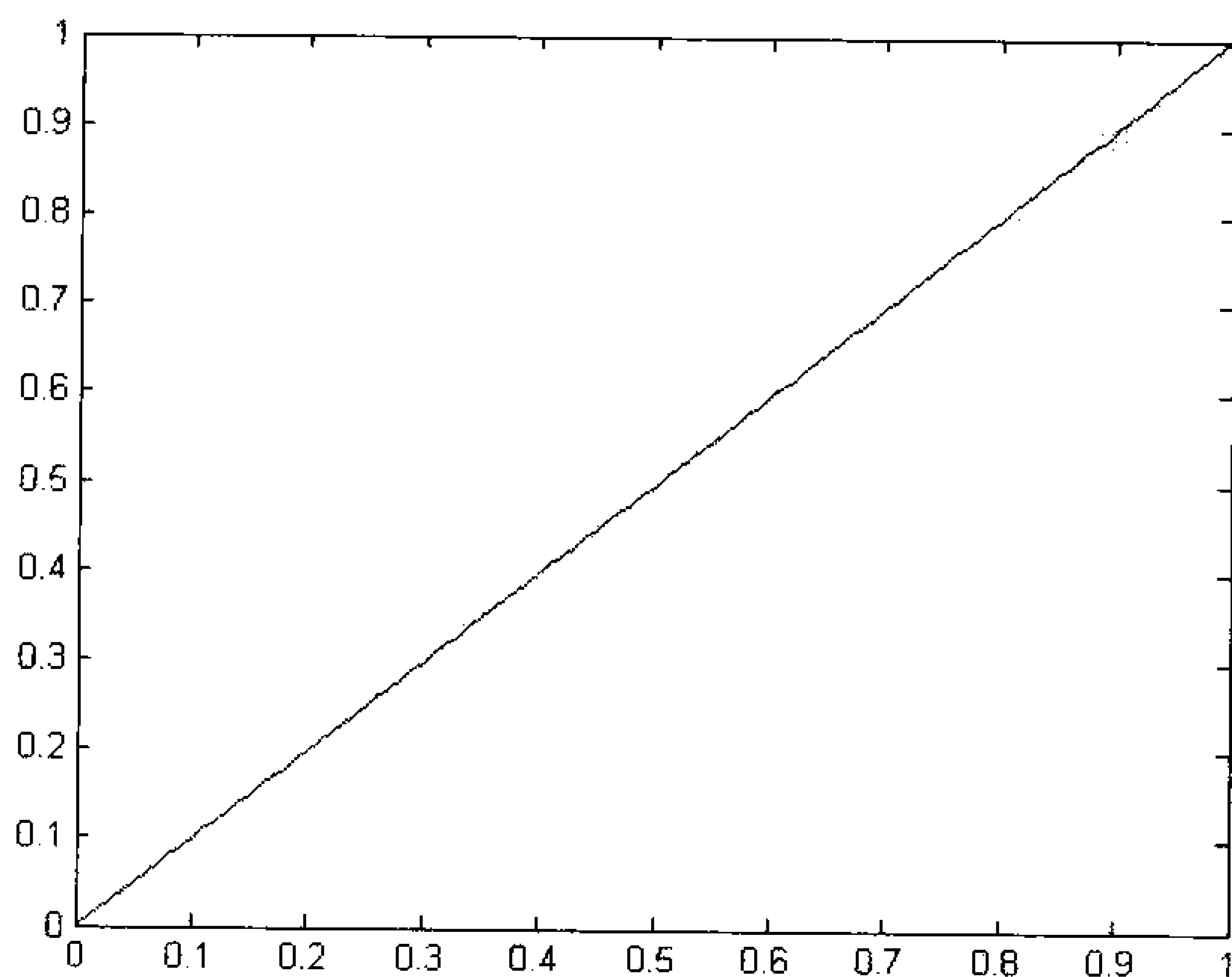


图 16-19 例 16-15 用克拉克-尼科尔森格式求得的当 t=0.5 时在求解域上的场函数值图

数值结果和隐式格式得出的结果是一样的。

16.3.1.5 加权隐式格式

加权隐式格式的形式如下所示：

$$\frac{u_j^{n+1}-u_j^n}{\tau}-\frac{a}{h^2}[\theta(u_{j+1}^{n+1}-2u_j^{n+1}+u_{j-1}^{n+1})+(1-\theta)(u_{j+1}^n-2u_j^n+u_{j-1}^n)]=0,\ (0<\theta<1)$$

其中 τ 为时间步长，h 为空间步长。

加权隐式格式用来求解如下形式的抛物问题：

$$\begin{cases}\dfrac{\partial u}{\partial t}=a\dfrac{\partial^2 u}{\partial x^2} & x\in[x_1,x_2],t>0\\ u(x_1,t)=u_1 & t>0\\ u(x_2,t)=u_2 & t>0\\ u(x,0)=U(x) & x\in[x_1,x_2]\end{cases}$$

在 MATLAB 中编程实现的扩散方程加权隐式格式的函数为：peParabWegImp

功能：用加权隐式格式解扩散方程的初值问题

调用格式：u = peParabWegImp (c,dt,n,minx,maxx,lbu,rbu,M)

其中，c：方程中的常数；
- dt：时间步长；
- n：空间节点个数；
- minx：求解区间的左端；
- maxx：求解区间的右端；
- lbu：左边界条件；
- rbu：右边界条件；
- M：时间步的个数；
- u：求解区间上的数值解。

加权隐式格式求解扩散方程初值问题的 MATLAB 程序代码如下所示：

```
function u = peParabWegImp(c,sita,dt,n,minx,maxx,lbu,rbu,M)
%方程中的常数：c
%时间步长：dt
%空间节点个数：n
%求解区间的左端：minx
%求解区间的右端：maxx
%左边界条件：lbu
%右边界条件：rbu
%时间步的个数：M
%求解区间上的数值解：u
format long;
h = (maxx-minx)/(n-1);
u0(1) = lbu;
u0(n) = rbu;
for j=2:n-1
```

```
        u0(j) = PrIniU(minx+(j-1)*h);
    end
    u1 = u0;
    for k=1:M
        A = zeros(n-2,n-2);
        cb =  zeros(n-2,1);
        cb(1) = -u0(2) -(1 - sita)*(u0(3)-2*u0(2)+u0(1))*dt*c*lbu/h/h/2 ...
            - sita*dt*c*lbu/h/h;
        cb(n-2) = -u0(n-1) -(1 -
sita)*(u0(n)-2*u0(n-1)+u0(n-2))*dt*c*lbu/h/h/2 ...
            - sita*dt*c*rbu/h/h;
        for i=2:n-3
            cb(i) = -u0(i+1) -(1 -
sita)*(u0(i+2)-2*u0(i+1)+u0(i))*dt*c*lbu/h/h;
        end
        A(1,1) = -2*sita*dt*c/h/h -1;
        A(1,2) = sita*dt*c/h/h ;
        for i=2:n-3
            A(i,i-1) = sita*dt*c/h/h ;
            A(i,i) = - 2*sita*dt*c/h/h -1 ;
            A(i,i+1) = sita*dt*c/h/h  ;
        end
        u1(2:(n-1)) = A\cb;
        u0 = u1;
    end
    u = u1;
    format short;
```

例 16-16 加权隐式格式求解扩散方程应用实例。用加权隐式格式求解下面扩散方程的初值问题：

$$\begin{cases} \dfrac{\partial u}{\partial t}=\dfrac{\partial^2 u}{\partial x^2} & -\infty < x < +\infty, t>0 \\ u(x,0)=\sin x & x\in(0,1) \\ u(0,t)=0 \\ u(1,t)=1 \end{cases}$$

其中时间步长取 0.005，空间步长取 0.01，求出当 t=0.5（即 100 个时间步）时的 u 随 x 的分布图，比较 θ 取不同值时的解。

解：先建立一个名为 PrIniU.m 的 MATLAB 文件，输入如下内容：

```
function ux = PrIniU(x)
format long;
ux = sin(x);
```

然后在 MATLAB 窗口输入下列命令：

```
>> u = peParabWegImp (1,0.5,0.005,101,0,1,0,1,100)
```

用加权隐式格式（$\theta=0.5$）求得的结果如图 16-20 所示：

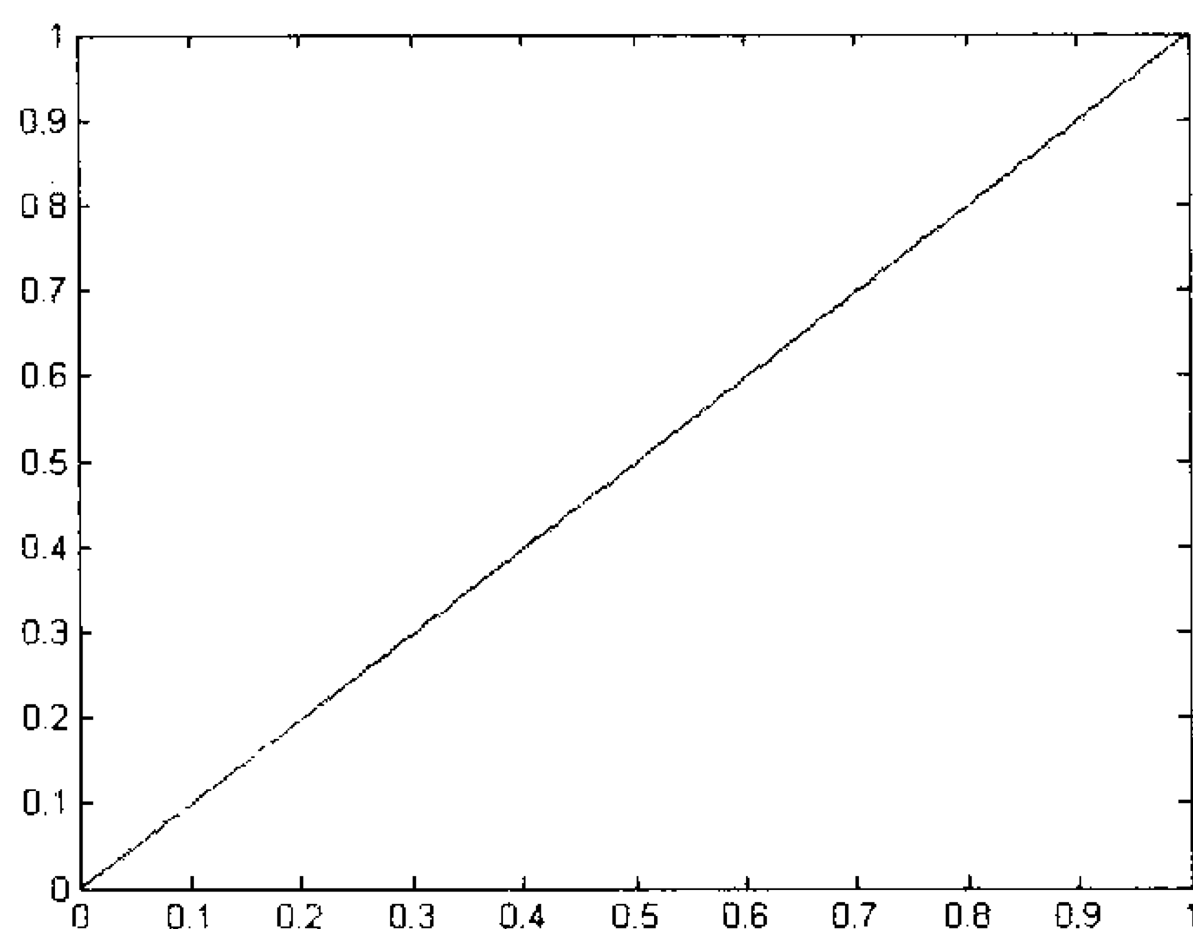


图 16-20　例 16-16 用加权隐式格式（ $\theta=0.5$ ）求得的当$\theta=0.5$时在求解域上的场函数值图

```
>> u = peParabWegImp (1,0.1,0.005,101,0.,1,0,1,100)
```

用加权隐式格式（ $\theta=0.1$ ）求得的结果如图 16-21 所示：

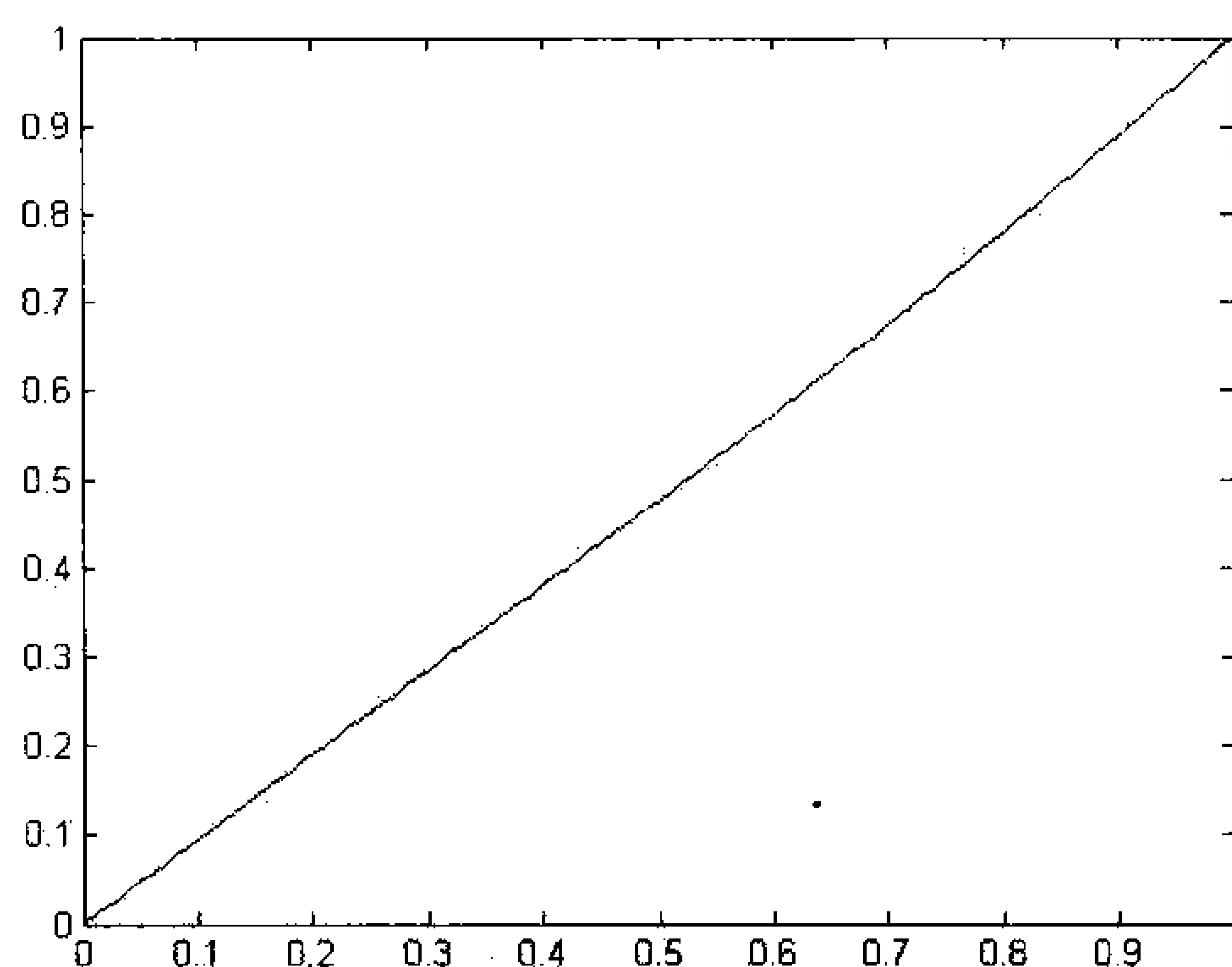


图 16-21　例 16-16 用加权隐式格式（ $\theta=0.1$ ）求得的当 $t=0.5$ 时在求解域上的场函数值图

```
>> u = peParabWegImp (1,0.9,0.005,101,0,1,0,1,100)
```

用加权隐式格式（ $\theta=0.9$ ）求得的结果如图 16.22 所示：

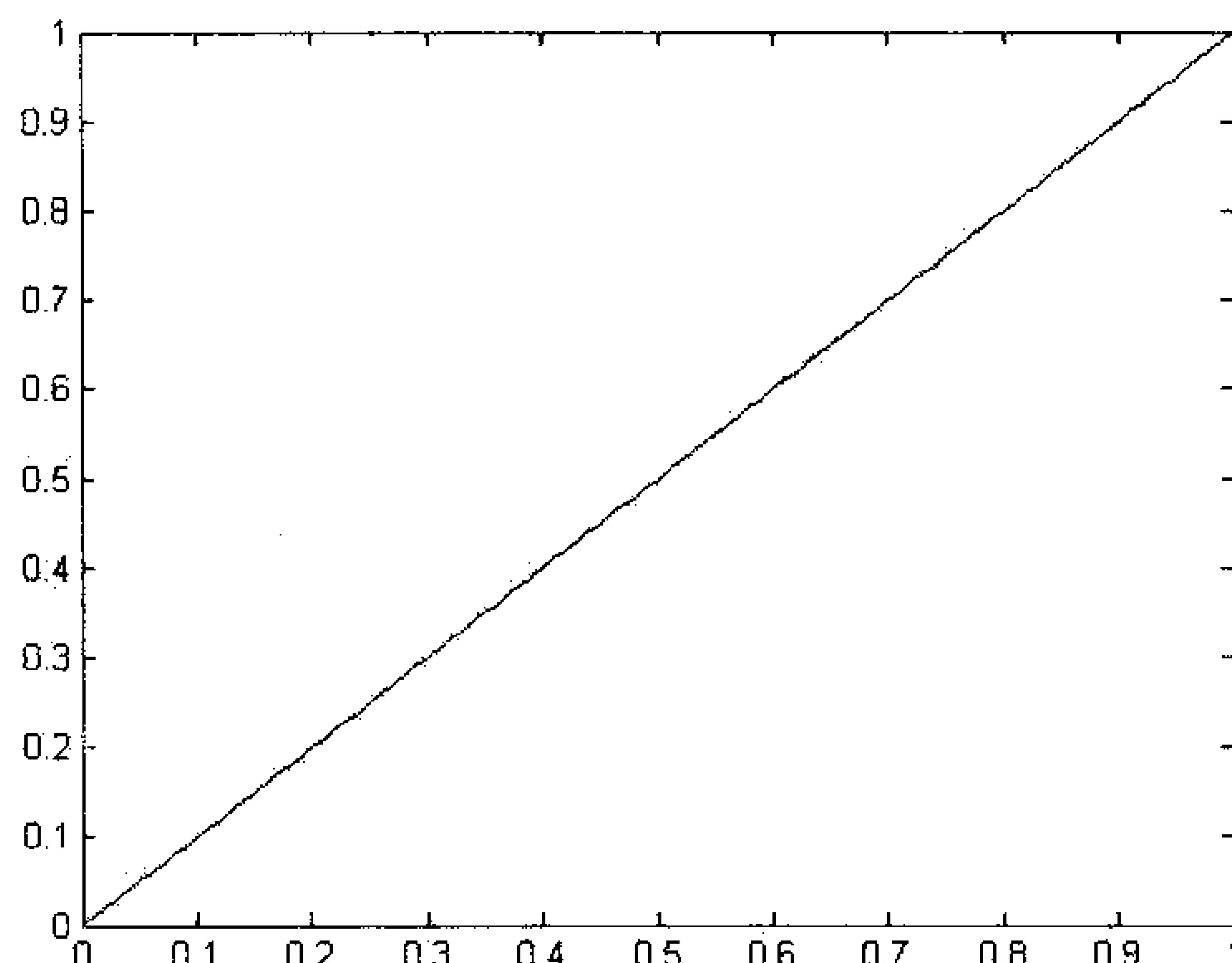


图 16-22　例 16-16 用加权隐式格式（ $\theta=0.9$ ）求得的当 $t=0.5$ 时在求解域上的场函数值图

例 16-16 的稳态解（即扩散达到稳定状态 $\dfrac{\partial u}{\partial t}=0$ ）为 $u=x$，通过取不同的θ值，用加权隐式格式算得的结果差别不大，只是达到稳态的时间不一样而已。

16.3.2　对流扩散方程

对流扩散方程是将对流方程和扩散方程组合在一起，其形式如下所示：

$$\frac{\partial u}{\partial t}+a\frac{\partial u}{\partial x}=b\frac{\partial^2 u}{\partial x^2}\quad x\in(-\infty,\infty),t\geqslant 0\ （a,b\text{ 为常数}）$$

将求解对流方程的差分格式和求解扩散方程的差分格式组合起来可以得到各种求解对流扩散方程的差分格式，下面介绍两种常用的指数型格式和萨马尔斯基格式。

16.3.2.1　指数型格式

指数型格式的形式如下所示：

$$\frac{u_j^{n+1}-u_j^n}{\tau}+\frac{a}{2h}(u_{j+1}^n-u_{j-1}^n)-\frac{ha}{2}\coth(\frac{ha}{2b})(u_{j+1}^n-2u_j^n+u_{j-1}^n)=0$$

其中 τ 为时间步长，h 为空间步长。

指数格式是一种显示格式，当然也需要进行节点延拓。

指数型格式用来求如下形式的抛物问题：

$$\begin{cases}\dfrac{\partial u}{\partial t}+a\dfrac{\partial u}{\partial x}=b\dfrac{\partial^2 u}{\partial x^2} & x\in(-\infty,\infty),t\geqslant 0\\ u(x,0)=U(x) & x\in(-\infty,\infty)\end{cases}$$

在 MATLAB 中编程实现的对流扩散方程指数型格式的函数为：peDKExp

功能：用指数型格式解对流扩散方程的初值问题

调用格式：u = peDKExp(a,b,dt,n,minx,maxx,M)

其中，a：方程中的常数 1；
　　b：方程中的常数 2；
　　dt：时间步长；
　　n：空间节点个数；
　　minx：求解区间的左端；
　　maxx：求解区间的右端；
　　M：时间步的个数；
　　u：求解区间上的数值解。

指数型格式求解对流扩散方程初值问题的 MATLAB 程序代码如下所示：

```
function u = peDKExp(a,b,dt,n,minx,maxx,M)
%方程中的常数 1：a
%方程中的常数 2：b
%时间步长：dt
%空间节点个数：n
%求解区间的左端：minx
%求解区间的右端：maxx
%时间步的个数：M
```

```
%求解区间上的数值解: u
format long;
h = (maxx-minx)/(n-1);
for j=1:(n+2*M)
    u0(j) = DKIniU(minx+(j-M-1)*h);          %节点延拓
end
u1 = u0;
coff = (exp(a*h/2/b)+exp(-a*h/2/b))/(exp(a*h/2/b)-exp(-a*h/2/b));
coff = dt*coff*a*h/2;
for k=1:M
    for i=k+1:n+2*M-k
        u1(i) =
coff*(u0(i+1)-2*u0(i)+u0(i-1))+a*dt*(u0(i+1)-u0(i-1))/h/2+u0(i);
    end
    u0 = u1;
end
u = u1((M+1):(M+n));
format short;
```

例 16-17 指数型格式求解对流扩散方程应用实例。用指数型格式求解下面对流扩散方程的初值问题:

$$\begin{cases} \dfrac{\partial u}{\partial t}+\dfrac{\partial u}{\partial x}=\dfrac{\partial^2 u}{\partial x^2} & x\in(-\infty,\infty),t\geqslant 0 \\ u(x,0)=x^2 & x\in(0,1) \end{cases}$$

其中时间步长取 0.005，空间步长取 0.01，求出当 t=0.5（即 100 个时间步）时的 u 随 x 的分布图。

解: 先建立一个名为 DKIniU.m 的 MATLAB 文件，输入如下内容:

```
function ux = DKIniU(x)
format long;
ux = x*x;
```

然后在 MATLAB 窗口输入下列命令:

```
>> u = peDKExp(1,1,0.005,101,0,1,100)
```

用指数型格式求得的结果如图 16-23 所示:

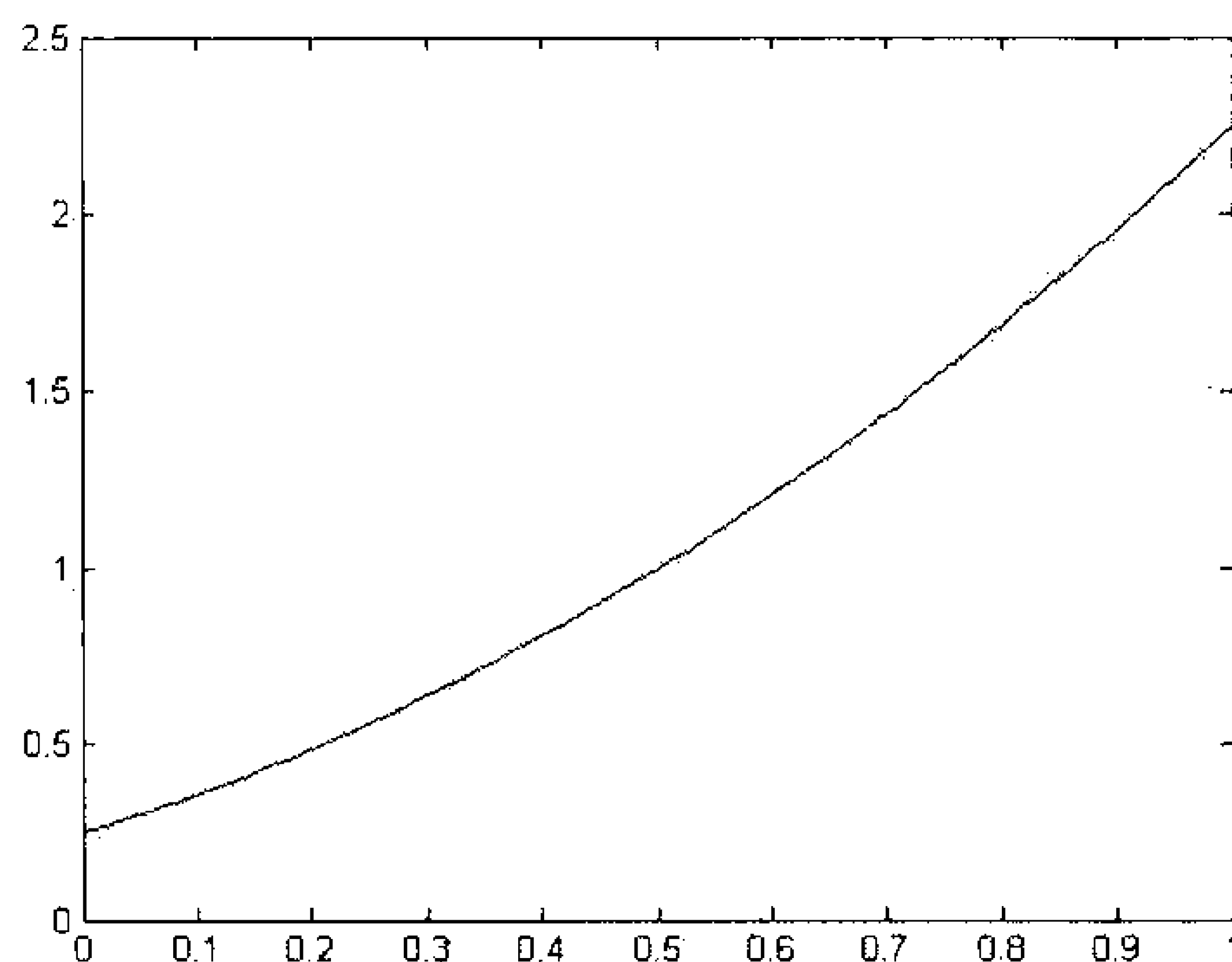


图 16-23 例 16-17 用指数型格式求得的当 $t=0.5$ 时在求解域上的场函数数值图

16.3.2.2　萨马尔斯基格式

萨马尔斯基格式的形式如下所示：

$$\frac{u_j^{n+1}-u_j^n}{\tau}+\frac{a}{h}(u_j^n-u_{j-1}^n)-\frac{b}{1+\dfrac{ah}{2b}}\frac{(u_{j+1}^n-2u_j^n+u_{j-1}^n)}{h^2}=0$$

其中τ为时间步长，h为空间步长。

萨马尔斯基格式用来求如下形式的抛物问题：

$$\begin{cases}\dfrac{\partial u}{\partial t}+a\dfrac{\partial u}{\partial x}=b\dfrac{\partial^2 u}{\partial x^2} & x\in(-\infty,\infty),t\geqslant 0\\ u(x,0)=U(x) & x\in(-\infty,\infty)\end{cases}$$

在 MATLAB 中编程实现的对流扩散方程萨马尔斯基格式的函数为：peDKSam

功能：用萨马尔斯基格式解对流扩散方程的初值问题

调用格式：u = peDKSam (a,b,dt,n,minx,maxx,M)

其中，a：方程中的常数 1；

b：方程中的常数 2；

dt：时间步长；

n：空间节点个数；

minx：求解区间的左端；

maxx：求解区间的右端；

M：时间步的个数；

u：求解区间上的数值解。

萨马尔斯基格式求解对流扩散方程初值问题的 MATLAB 程序代码如下所示：

```
function u = peDKSam(a,b,dt,n,minx,maxx,M)
%方程中的常数 1：a
%方程中的常数 2：b
%时间步长：dt
%空间节点个数：n
%求解区间的左端：minx
%求解区间的右端：maxx
%时间步的个数：M
%求解区间上的数值解：u
format long;
h = (maxx-minx)/(n-1);
for j=1:(n+2*M)
    u0(j) = DKIniU(minx+(j-M-1)*h);
end
u1 = u0;
coff = dt*b/(1+a*h/b/2);
for k=1:M
    for i=k+1:n+2*M-k
```

```
            u1(i) =
coff*(u0(i+1)-2*u0(i)+u0(i-1))+a*dt*(u0(i)-u0(i-1))/h+u0(i);
        end
        u0 = u1;
    end
    u = u1((M+1):(M+n));
    format short;
```

例 16-18 萨马尔斯基格式求解对流扩散方程应用实例。用萨马尔斯基格式求解下面对流扩散方程的初值问题：

$$\begin{cases} \dfrac{\partial u}{\partial t}+\dfrac{\partial u}{\partial x}=\dfrac{\partial^2 u}{\partial x^2} & x\in(-\infty,\infty),t\geqslant 0 \\ u(x,0)=x^2 & x\in(0,1) \end{cases}$$

其中时间步长取 0.005，空间步长取 0.01，求出当 $t=0.5$（即 100 个时间步）时的 u 随 x 的分布图。

解：先建立一个名为 DKIniU.m 的 MATLAB 文件，输入如下内容：

```
function ux = DKIniU(x)
format long;
ux = x*x;
```

然后在 MATLAB 窗口输入下列命令：

```
>> u = peDKSam(1,1,0.005,101,0,1,100)
```

用萨马尔斯基格式求得的结果如图 16-24 所示：

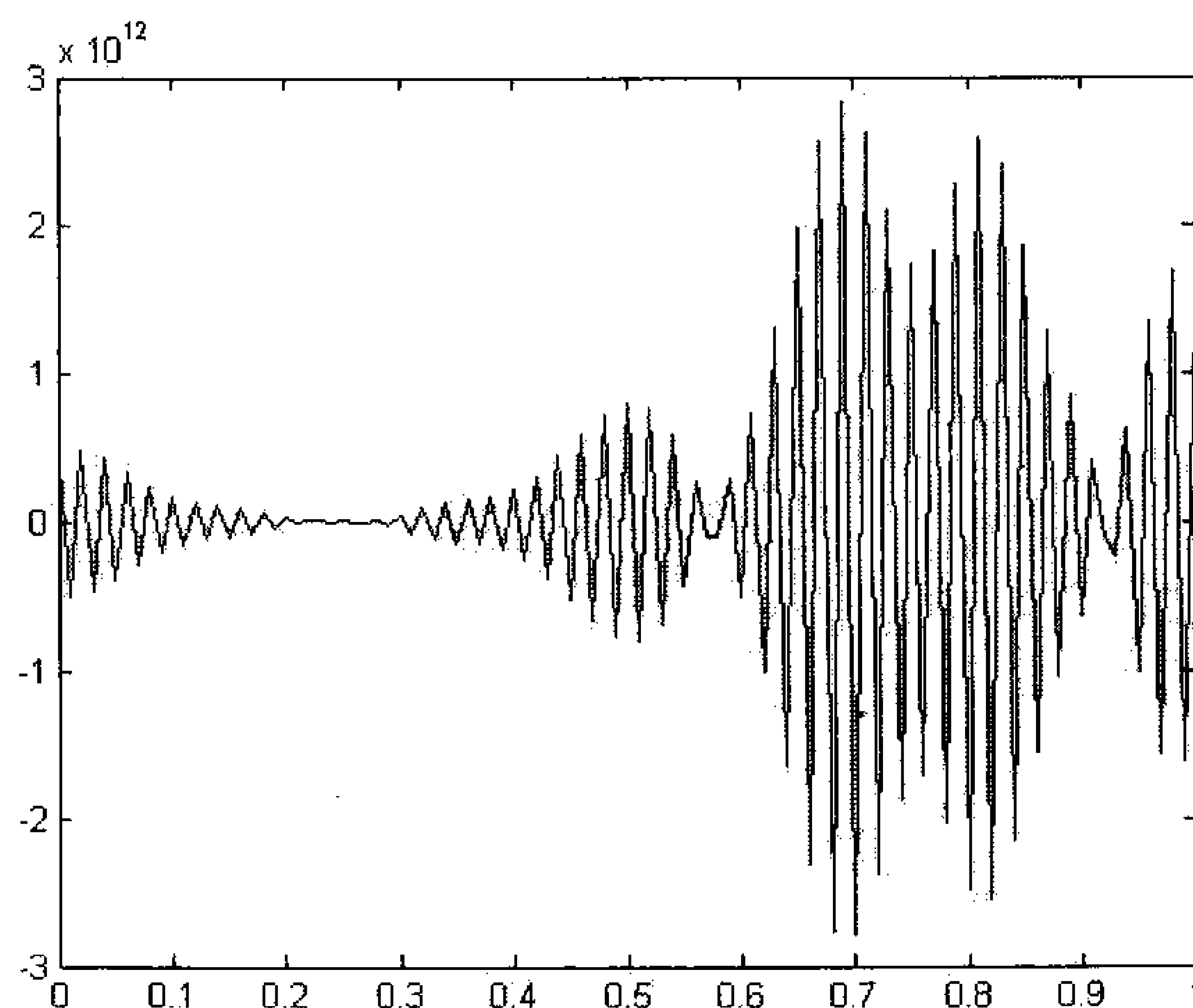


图 16-24　例 16-18 用萨马尔斯基格式求得的当 $t=0.5$ 时在求解域上的场函数值图

```
>> u = peDKSam(1,1,0.0001,101,0,1,1000)
```

改变参数后求得的结果如图 16-25 所示：

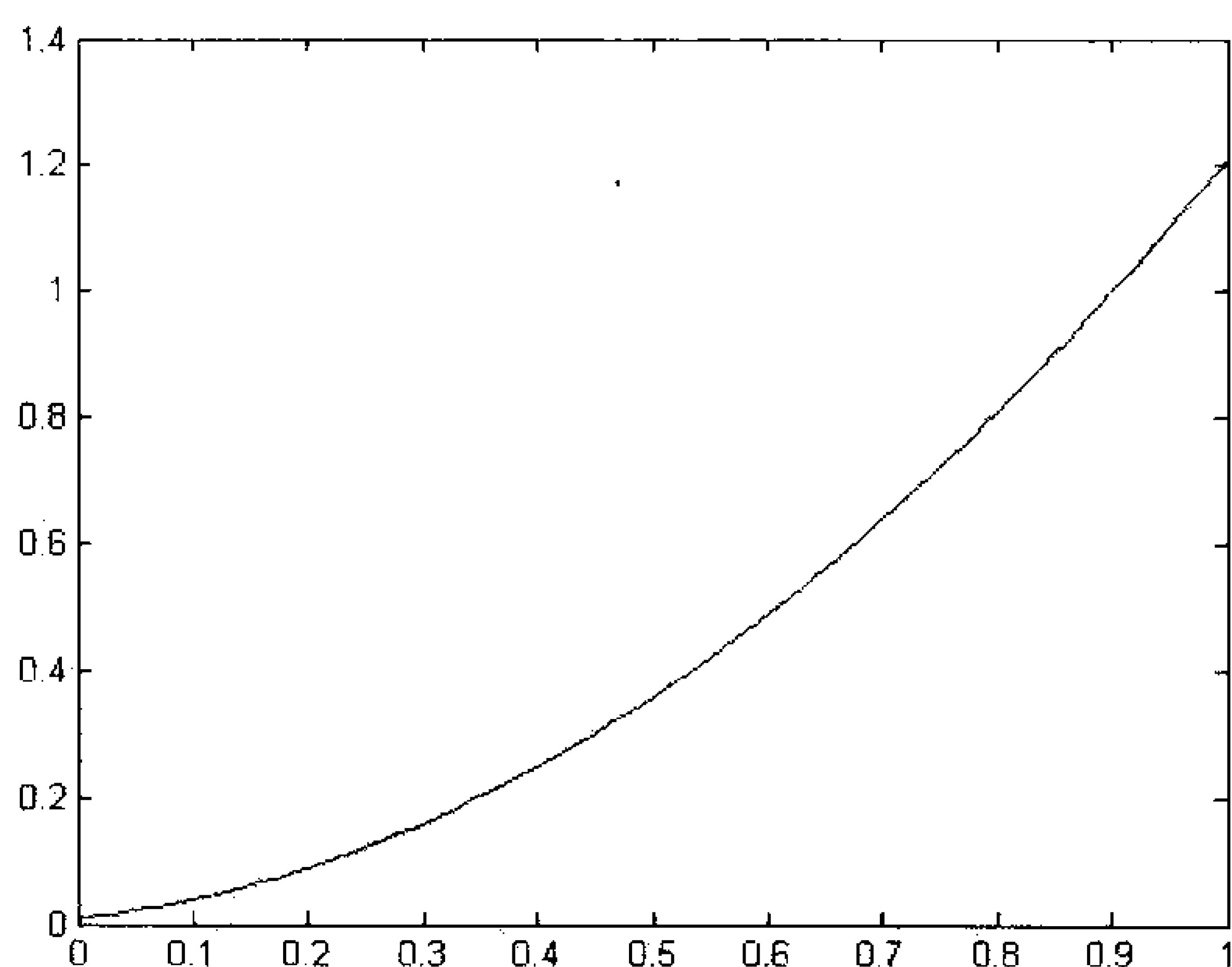


图 16-25　例 16-18 用改变参数后的萨马尔斯基格式求得的在求解域上的场函数数值图

这时的数值结果没有发生振荡，得到了正确的结果，所以对于各种条件稳定的差分格式要注意其稳定性条件才能得出正确的结果。

16.4 小结

本章分别介绍了差分方法在椭圆型、抛物型和双曲型偏微分方程中的应用，用差分格式求解偏微分方程的步骤基本是一样的，首先把连续问题离散化，建立差分格式，然后根据差分格式对求解区域进行网格剖分，最后求解方程。

除了差分法外，变分法和有限元法也是常用的方法，另外随着计算方法的发展，还出现了边界元法、混合有限元法和多重网格法等新的方法，读者如果有兴趣的话，可参考相关的书籍。

第 17 章　数据统计和分析

研究自然界中某些变量之间的关系时，往往是从分析变量的观察值着手，这些观察值称为样本，要研究的变量全体取值称为总体。数据的统计和分析就是通过分析样本的主要特征，如分布、趋势、集中程度等来探究变量之间的内在关系，并且预测某些变量的发展趋势。

通过本章的学习，读者不仅能了解和掌握常见的数据统计和分析方法，而且还能熟练使用 MATLAB 编程来实现这些算法。

17.1 回归分析

回归分析是应用极其广泛的一种数据分析方法，它通过揭示变量之间内在的关系来反映某种规律，从而预测或控制感兴趣的一个或几个变量的变化。

按照变量之间的关系，可以把回归分析分为线性回归和非线性回归，实验数据的直线拟合其实就是一个线性回归的例子，而二次拟合就是一个简单的非线性回归的例子。

17.1.1 线性回归

线性回归的模型为：$y=\beta_0+\beta_1x_1+\cdots+\beta_nx_n$，在得到自变量的 N 组观测值 X 后，

$$X=\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{Np} \end{bmatrix}$$

对其进行线性回归的步骤介绍如下。

❶ 输入原始数据。

$$X=\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{Nn} \end{bmatrix},\quad Y=\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

其中 n 为自变量个数，N 为数据组数。

❷ 计算 $\overline{X}=\begin{bmatrix}\overline{x_1} & \overline{x_2} & \cdots & \overline{x_n}\end{bmatrix}^T$，其中 $\overline{x_i}=\frac{1}{N}\sum_{j=1}^{N}x_{ji}$。

❸ 计算$\overline{Y}=\frac{1}{N}\sum_{j=1}^{N}y_j$。

❹ 解方程$A\beta=C$，其中

$$A=X^TX-N\overline{X}\overline{X}^T, C=X^TY-N\overline{X}\overline{Y}, \beta=[\beta_1 \quad \beta_2 \quad \cdots \quad \beta_n]。$$

❺ 计算$\beta_0=\overline{Y}-\overline{X}^T\beta$。

❻ 进行显著性检验。

总离差平方和：$S=\sum_{i=1}^{N}(y_i-\overline{Y})^2$；

回归平方和：$U=\sum_{i=1}^{N}(\widehat{y_i}-\overline{Y})^2$，其中$\widehat{y_i}=\beta_0+\beta_1x_{i1}+\cdots+\beta_nx_{in}$；

剩余平方和：$Q=S-U$；

复可决系数：$R^2=\frac{U}{S}$；

复相关系数：$R=\sqrt{\frac{U}{S}}$；

回归均方：$\overline{U}=\frac{U}{n}$；

剩余均方：$\overline{Q}=\frac{Q}{N-n-1}$；

剩余标准差：$s=\sqrt{\overline{Q}}$；

方程显著性检验值：$F=\frac{\overline{U}}{\overline{Q}}$。

根据F和$F(x_j)$的值判断各个因子的显著性。

① F检验值$F(x_j)$：$F(x_j)=\frac{\beta_j^2}{l_{jj}\overline{Q}}$，其中$[l_{ij}]=(X^TX-N*\overline{X}\overline{X}^T)^{-1}$；在给定$\alpha$后，如果$F(x_j)\geqslant F_\alpha(1,N-n-1)$，则因子$x_j$显著，否则$x_j$不显著。

② t检验值$t(x_j)$：$t(x_j)=\frac{\beta_j}{\sqrt{l_{jj}}s}$，在给定$\alpha$后，如果$|t(x_j)|>t_{\frac{\alpha}{2}}(N-n-1)$，则因子$x_j$显著，否则$x_j$不显著。

回归系数的标准差：

$$s_{\beta_0}=\sqrt{\overline{Q}(\frac{1}{N}+\overline{X}^TA^{-1}\overline{X})}$$

$$s_{\beta_i}=\sqrt{\overline{Q}l_{ii}}$$

❼ 输出回归方程：

$$y=\beta_0+\beta_1x_1+\cdots+\beta_nx_n$$

在 MATLAB 中编程实现的线性回归法的函数为：MultiLineReg

功能：用线性回归法估计一个因变量与多个自变量之间的线性关系

调用格式：[RegCoff,R,F,FX,TX]= MultiLineReg(X,Y)

其中，X：自变量的样本矩阵；

Y：因变量样本矩阵；

RegCoff：线性回归系数；

R：复相关系数；

F：方程显著性检验 F 值；

FX：各因子显著性检验 F 值；

TX：各因子显著性检验 t 值。

线性回归法的 MATLAB 程序代码如下所示：

```
function [RegCoff,R,F,FX,TX]= MultiLineReg(X,Y)
%自变量的样本矩阵：X
%因变量样本矩阵：Y
%线性回归系数：RegCoff
%复相关系数：R
%方程显著性检验 F 值：F
%各因子显著性检验 F 值：FX
%各因子显著性检验 t 值：TX
format long;
sz = size(X);
N = sz(1);
n = sz(2);
RegCoff = zeros(n+1,1);              %回归系数
Z = mean(X);
yp = mean(Y);
A = transpose(X)*X - N*transpose(Z)*Z;  %A 矩阵
C = transpose(X)*Y - N*transpose(Z)*yp; %C 矩阵
RegCoff(2:n+1) = A\C;
RegCoff(1) = yp - Z*RegCoff(2:n+1);  %回归系数的常数项
S = norm(Y)^2 - N*yp^2;              %显著性检验
YR = X*RegCoff(2:n+1) + RegCoff(1)*ones(N,1);
U = transpose(RegCoff(2:n+1))*C;
Q = S-U;
R = sqrt(U/S);
UR = U/(length(RegCoff)-1);
QR = Q/(N -length(RegCoff));
s = sqrt(QR);
inA = inv(A);
F = UR/QR;
for i=1:length(RegCoff)-1      %因子显著性检验
    FX(i) = RegCoff(i+1)^2/inA(i,i)/QR;
    TX(i) = RegCoff(i+1)/sqrt(inA(i,i)*s);
end
format short;
```

例 17-1 线性回归法应用实例。用线性回归法估计下表中的 y 与自变量 x_1, x_2, x_3 的

线性关系。

y	x_1	x_2	x_3
160	70	35	1.0
260	75	40	2.4
210	65	40	2.0
265	74	42	3.0
240	72	38	1.2
220	68	45	1.5
275	78	42	4.0
160	66	36	2.0
275	70	44	3.2
250	65	42	3.0

解：先建立一个数据文件 XData.txt，并输入下列 10 行 3 列的数据：

```
70    35    1.0
75    40    2.4
65    40    2.0
74    42    3.0
72    38    1.2
68    45    1.5
78    42    4.0
66    36    2.0
70    44    3.2
65    42    3.0
```

再建立一个数据文件 YData.txt，并输入下列数据：

```
160
260
210
265
240
220
275
160
275
250
```

解：在 MATLAB 窗口输入下列命令：

```
>> X=load('XData.txt');
>> Y=load('YData.txt');
>> [RegCoff,R,F,FX,TX]= MultiLineReg(X,Y)
RegCoff =
 -348.2802
    3.7540
    7.1007
   12.4475
```

```
R =  0.8975
F =  8.2832
FX = 3.7704    6.0776    1.3870
TX = 1.9418    2.4653    1.1777
```

从结果可以看出，回归模型为：

$$y = -348.2802 + 3.754x_1 + 7.1007x_2 + 12.4475x_3$$

取 $\alpha = 0.05$ 对方程和回归系数进行检验，查 F 分布表可得 $F_{0.05}(3,6) = 4.76$，本例中的方程检验值 $F = 8.2832 > 4.76$，说明方程显著；查 F 分布表可得 $F_{0.05}(1,6) = 5.99$，

$F_1 = 3.7704 < 5.99$，说明 x_1 不显著；

$F_2 = 6.0776 > 5.99$，说明 x_2 显著；

$F_3 = 1.3870 < 5.99$，说明 x_3 不显著。

17.1.2 多项式回归

多项式回归是一种非线性回归（大于一次的多项式），它研究两个变量之间的多项式关系，其模型为：

$$y = \beta_0 + \beta_1 x + \cdots + \beta_n x^n$$

在得到自变量的 N 组观测值后，将输入数据改写为：

$$X = \begin{bmatrix} x_1 & x_1^2 & \cdots & x_1^n \\ x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \ddots & \vdots \\ x_N & x_N^2 & \cdots & x_N^n \end{bmatrix}, \quad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

后面的过程同第 17.1.1 节的线性回归完全一样。

在 MATLAB 中编程实现的多项式回归的函数为：PolyReg

功能：用多项式回归法估计一个因变量与一个自变量之间的多项式关系

调用格式：[RegCoff,R,F,FX,TX]= PolyReg (X,Y,n)

其中，X：自变量的样本矩阵；

Y：因变量样本矩阵；

n：多项式的幂次；

RegCoff：多项式回归系数；

R：复相关系数；

F：方程显著性检验 F 值；

FX：各因子显著性检验 F 值；

TX：各因子显著性检验 t 值。

多项式回归法的 MATLAB 程序代码如下所示：

```
function [RegCoff,R,F,FX,TX]= PolyReg(X,Y,n)
%自变量的样本矩阵：X
```

```
%因变量样本矩阵：Y
%多项式的幂次：n
%多项式回归系数：RegCoff
%复相关系数：R
%方程显著性检验 F 值：F
%各因子显著性检验 F 值：FX
%各因子显著性检验 t 值：TX
function [RegCoff,R,F,FX,TX]= PolyReg(X,Y,n)
format long;
sz = size(X);
N = sz(1);
XX = zeros(N,n);
for i=1:N
    for j=1:n
        XX(i,j) = X(i)^j;
    end
end                     %输入矩阵
RegCoff = zeros(n+1,1);
Z = mean(XX);
yp = mean(Y);
A = transpose(XX)*XX - N*transpose(Z)*Z;
C = transpose(XX)*Y - N*transpose(Z)*yp;
RegCoff(2:n+1) = A\C;
RegCoff(1) = yp - Z*RegCoff(2:n+1);
S = norm(Y)^2 - N*yp^2;        %显著性检验
YR = XX*RegCoff(2:n+1) + RegCoff(1)*ones(N,1);
U = transpose(RegCoff(2:n+1))*C;
Q = S-U;
R = sqrt(U/S);
UR = U/n;
QR = Q/(N-n-1);
s = sqrt(QR);
inA = inv(A);
F = UR/QR;
for i=1:length(RegCoff)-1   %因子显著性检验
   FX(i) = RegCoff(i+1)^2/inA(i,i)/QR;
   TX(i) = RegCoff(i+1)/sqrt(inA(i,i))/s;
end
format short;
```

例 17-2 多项式回归法应用实例。用三次多项式回归法估计下表中的 y 与自变量 x 的线性关系。

y	x
160	35
260	40
210	40
265	42

续表

y	x
240	38
220	45
275	42
160	36
275	44
250	42

解： 先建立一个数据文件 Xdata1.txt，并输入下列数据：

```
35
40
40
42
38
45
42
36
44
42
```

再建立一个数据文件 Ydata1.txt，并输入下列数据：

```
160
260
210
265
240
220
275
160
275
250
```

在 MATLAB 窗口输入下列命令：

```
>> X=load('Xdata1.txt');
>> Y=load('Ydata1.txt');
>> [RegCoff,R,F,FX,TX]= PolyReg(X,Y,3)
RegCoff =
  1.0e+004 *
    1.2613
   -0.1039
    0.0028
   -0.0000
R =  0.8925
F =  7.8312
FX =  0.4198    0.5023    0.5829
TX = -0.6479    0.7087   -0.7635
```

从分析结果可以得到 y 与 x 之间的三次多项式回归模型为：

$$y = 12613 - 1039x + 28x^2 + 0 \cdot x^3$$

注意三次项系数为 0，说明三次项非常不显著。

17.1.3 二次完全式回归

二次完全式回归的模型为：

$$y = \beta_0 + \sum_{i=1}^{2} \beta_{i0} x_i + \frac{1}{2} \sum_{i=1}^{2} \sum_{j=1}^{2} \beta_{ij} x_i x_j, (\beta_{ij} = \beta_{ji})$$

在得到自变量的 N 组观测值后，将输入数据改写为：

$$X = \begin{bmatrix} x_{11} & {x_{11}}^2 & x_{11}x_{12} & x_{12} & {x_{12}}^2 \\ x_{21} & {x_{21}}^2 & x_{21}x_{22} & x_{22} & {x_{22}}^2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{N1} & {x_{N1}}^2 & x_{N1}x_{N2} & x_{N2} & {x_{N2}}^2 \end{bmatrix}, \quad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

后面的过程同线性回归完全一样。

在 MATLAB 中编程实现的二次完全式回归的函数为：CompPoly2Reg
功能：用二次完全式回归法估计一个因变量与两个自变量之间的关系
调用格式：[RegCoff,R,F,FX,TX]= CompPoly2Reg (X,Y)
其中，X：自变量的样本矩阵；
Y：因变量样本矩阵；
RegCoff：二次完全式回归系数；
R：复相关系数；
F：方程显著性检验 F 值；
FX：各因子显著性检验 F 值；
TX：各因子显著性检验 t 值。

二次完全式回归法的 MATLAB 程序代码如下所示：

```
function [RegCoff,R,F,FX,TX]= CompPoly2Reg(X,Y)
%自变量的样本矩阵：X
%因变量样本矩阵：Y
%二次完全式回归系数：RegCoff
%复相关系数：R
%方程显著性检验 F 值：F
%各因子显著性检验 F 值：FX
%各因子显著性检验 t 值：TX
format long;
sz = size(X);
N = sz(1);
n = sz(2);
XX = zeros(N,n);
```

```
XX(:,1) = X(:,1);
XX(:,4) = X(:,2);
for i=1:N
    XX(i,2) = X(i,1)^2;
    XX(i,3) = X(i,1)*X(i,2);
    XX(i,5) = X(i,2)^2;
End                       %输入矩阵
RegCoff = zeros(6,1);
Z = mean(XX);
yp = mean(Y);
A = transpose(XX)*XX - N*transpose(Z)*Z;
C = transpose(XX)*Y - N*transpose(Z)*yp;
RegCoff(2:6) = A\C;
RegCoff(1) = yp - Z*RegCoff(2:6);
S = norm(Y)^2 - N*yp^2;    %显著性检验
YR = XX*RegCoff(2:6) + RegCoff(1)*ones(N,1);
U = transpose(RegCoff(2:6))*C;
Q = S-U;
R = sqrt(U/S);
UR = U/n;
QR = Q/(N-6);
s = sqrt(QR);
inA = inv(A);
F = UR/QR;
for i=1:length(RegCoff)-1   %因子显著性检验
    FX(i) = RegCoff(i+1)^2/inA(i,i)/QR;
    TX(i) = RegCoff(i+1)/sqrt(inA(i,i))/s;
end
format short;
```

例 17-3 二次完全式回归法应用实例。用二次完全式回归法估计下表中的 y 与自变量 x_1 及 x_2 的线性关系。

y	x_1	x_2
160	70	35
260	75	40
210	65	40
265	74	42
240	72	38
220	68	45
275	78	42
160	66	36
275	70	44
250	65	42

解： 先建立一个数据文件 Xdata2.txt，并输入下列数据：

```
70    35
```

```
75    40
65    40
74    42
72    38
68    45
78    42
66    36
70    44
65    42
```

再建立一个数据文件 Ydata2.txt，并输入下列数据：

```
160
260
210
265
240
220
275
160
275
250
```

在 MATLAB 窗口输入下列命令：

```
>> X=load('Xdata2.txt');
>> Y=load('Ydata2.txt');
>> [RegCoff,R,F,FX,TX]= CompPoly2Reg(X,Y)
RegCoff =
  1.0e+003 *
   -6.1465
    0.0654
   -0.0004
   -0.0002
    0.1895
   -0.0021
R =  0.9628
F =  25.3877
FX = 1.3425    0.6766    0.0285    6.5254    7.8871
TX = 1.1586   -0.8226   -0.1687    2.5545   -2.8084
```

从分析结果可以得到 y 与自变量 x_1,x_2 之间的二次完全回归模型为：

$$y = -6146.5 + 65.4x_1 - 0.4x_1^2 - 0.2x_1x_2 + 189.5x_2 - 2.1x_2^2$$

17.2 聚类分析

聚类分析本质上是将研究的对象进行分类，其基本思想是通过定义样本之间的距离，将相近的样本归为一类，直至所有类之间的距离满足某种条件。聚类分析中常用的是系统聚类法。

系统聚类法的算法过程介绍如下。

❶ 计算 n 个样本两两之间的距离，形成距离矩阵 D，D 为对称矩阵，且对角元素为 0。

❷ 首先构造 n 个类，每一类中只包含一个样本。

❸ 合并距离最近的两类为新类。

❹ 计算新类与当前各类的距离，若类的个数已经等于 1，转❺，否则转❸。

❺ 决定类的个数和类。

类的距离可以采用以下两种算法：

① 最短距离法：

$$D(C_1,C_2)=\min_{\substack{x_i\in C_1\\x_j\in C_2}}\{d(x_i,x_j)\}$$

② 最长距离法：

$$D(C_1,C_2)=\max_{\substack{x_i\in C_1\\x_j\in C_2}}\{d(x_i,x_j)\}$$

其中 $d(x_i,x_j)=\sqrt{\sum_{k=1}^{m}(x_{ik}-x_{jk})^2)}$，为两个类中任意两个样本的距离。

> 在 MATLAB 中编程实现的系统聚类法的函数为：CollectAnaly
>
> 功能：用最短距离算法的系统聚类对样本进行聚类
>
> 调用格式：CollectAnaly(X)
>
> 其中，X：样本矩阵。

系统聚类法的 MATLAB 程序代码如下所示：

```
function CollectAnaly(X)
%样本矩阵：X
format long;
sz = size(X);
N = sz(1);          %样本个数
n = sz(2);
D = zeros(n,n);
totalClass = N;
RecordClass = zeros(N,N+1);   %构造 n 个类
RecordClass(:,1) = ones(N,1);
RecordClass(:,2) = 1:N;
disp('聚类前的 N 个类：');
disp(RecordClass);
while totalClass > 1          %合并距离最近的两类
    minClaDist = inf;
    for i=1:totalClass
        for j=i+1:totalClass
            distClass = DistClass(X,RecordClass,i,j,RecordClass(i,1),
RecordClass(j,1));  %类的距离
            if distClass < minClaDist      %找出距离最近的两类
                minClaDist = distClass;
                i1 = i;
                j1 = j;
```

```
            end
        end
    end
    %合并距离最近的两类
    t1 = RecordClass(i1,1);
    t2 = RecordClass(j1,1);
    RecordClass(i1,(t1+2):(t1+t2+1)) = RecordClass(j1,2:(t2+1));
    RecordClass(i1,1) = RecordClass(i1,1) + RecordClass(j1,1);
    RecordClass(j1:(totalClass-1),:) = RecordClass((j1+1):totalClass,:);
    RecordClass(totalClass:N,:) = zeros(N-totalClass+1,N+1);
    totalClass = totalClass - 1;
    str1 = strcat('第',num2str(N - totalClass));
    str1 = strcat(str1, '次聚类, ');
    str1 = strcat(str1, '第');
    str1 = strcat(str1, num2str(i1));
    str1 = strcat(str1, '类和第');
    str1 = strcat(str1, num2str(j1));
    str1 = strcat(str1, '类合并: ');
    disp(str1);
    disp(RecordClass(1:totalClass,:))
end
function d = DistSamp(X1,X2)     %样本距离的计算函数
format long;
d = sqrt(dot( X1-X2,X1-X2));
format short;
function d = DistClass(X,C1,l1,l2,n,m) %类距离的计算函数
format long;
d = inf;
for i=1:n
    for j=1:m
        dc = DistSamp(X(C1(l1,i+1),:),X(C1(l2,j+1),:));
        if dc < d
            d = dc;
        end
    end
end
format short;
```

例 17-4 最短距离算法的系统聚类应用实例。对下面的 5 个样本用最短距离法进行分类。

样本序号	x_1	x_2
1	1	1
2	1	2
3	6	3
4	8	2
5	8	0

解： 在 MATLAB 命令窗口输入下列命令：

```
>> X = [ 1    1;1    2;6    3;8    2;8    0];
>> CollectAnaly(X)
%聚类前的 N 个类:
     1     1     0     0     0     0
     1     2     0     0     0     0
     1     3     0     0     0     0
     1     4     0     0     0     0
     1     5     0     0     0     0
%第 1 次聚类，第 1 类和第 2 类合并:
     2     1     2     0     0     0
     1     3     0     0     0     0
     1     4     0     0     0     0
     1     5     0     0     0     0
%第 2 次聚类，第 3 类和第 4 类合并:
     2     1     2     0     0     0
     1     3     0     0     0     0
     2     4     5     0     0     0
%第 3 次聚类，第 2 类和第 3 类合并:
     2     1     2     0     0     0
     3     3     4     5     0     0
%第 4 次聚类，第 1 类和第 2 类合并:
     5     1     2     3     4     5
```

输出数据的行数代表每次聚类后类的个数，每一行的第一个数表示此类中样本的个数，后面的数代表此类中的样本编号；从程序的输出结果看，聚类的过程一目了然，未聚类前，有 5 个类，每个类各有一个样本；第一次聚类是将第一类和第二类合并，得到新的第一类，此时第一类有 2 个样本；而第二次聚类是将第 3 类和第 4 类合并，得到新的第三类；第 3 次聚类是将第 2 类和第 3 类合并，此时总共有 2 类，经过第四次聚类，类的个数变为 1，聚类的过程到此完成。

17.3 判别分析

判别分析也是一种分类法，但是和聚类分析不同的是，判别分析是在分类已经给定的情况下，通过某种判别法则，判断新的样本属于哪个已知类。

下面介绍比较简单的 Fisher 两类判别算法，其思想是已经给定两类的训练样本，判断新的样本属于哪类。

Fisher 两类判别算法介绍如下。

❶ 输入两类样本值：

$$X(A)=\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{M1} & x_{M2} & \cdots & x_{Mp} \end{bmatrix}$$

$$X(B)=\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{Np} \end{bmatrix}$$

❷ 计算均值：

$$\begin{cases} \overline{X}_l(A)=\dfrac{1}{M}\sum_{i=1}^{M}x_{il}(A) \\ \overline{X}_l(B)=\dfrac{1}{N}\sum_{i=1}^{N}x_{il}(B) \end{cases} \quad (l=1,2,\cdots,p)$$

计算方差：

$$S_{jk}=\sum_{i=1}^{M}[x_{ij}(A)-\overline{X}_j(A)][x_{ik}(A)-\overline{X}_k(A)] + \sum_{i=1}^{N}[x_{ij}(B)-\overline{X}_j(B)][x_{ik}(B)-\overline{X}_k(B)]$$

$$(j=1,2,\cdots,p;\ k=1,2,\cdots,p)$$

计算距离：

$$d_l=\overline{X}_l(A)-\overline{X}_l(B) \qquad (l=1,2,\cdots,p)$$

❸ 解方程组 $Sc=d$ 求出判别系数 c，建立判别函数

$$Y=\sum_{i=1}^{p}c_p x_i$$

❹ 用判别函数判别新样本的类别。

在 MATLAB 中编程实现的 Fisher 两类判别法的函数为：DistgshAnalysis

功能：用 Fisher 两类判别法对样本进行分类

调用格式：kindX = DistgshAnalysis(XA,XB,SampX)

其中，XA：第一类的样本矩阵；

XB：第二类的样本矩阵；

SampX：需要分类的样本；

kindX：样本的类别。

Fisher 两类判别法的 MATLAB 程序代码如下所示：

```
function kindX = DistgshAnalysis(XA,XB,SampX)
%第一类的样本矩阵：XA
%第二类的样本矩阵：XB
%需要分类的样本：SampX
%样本的类别：kindX
format long;
```

```
    sz1 = size(XA);
    sz2 = size(XB);
    M = sz1(1);            %样本个数
    N = sz2(1);
    n = sz1(2);
    meanXA = mean(XA);
    meanXB = mean(XB);
    sx = zeros(n,n);
    Y = zeros(N,n);
    for i=1:n
        for j=1:n
            sx(i,j) = dot(XA(:,i)-meanXA(i)*zeros(M,1),XA(:,j)-meanXA(j)*zeros
(M,1))+ ...
                 dot(XB(:,i)-meanXB(i)*zeros(N,1),XB(:,j)-meanXB(j)*zeros
(N,1));
        end
    end
    d = transpose(meanXA - meanXB);
    c = sx\d;            %判别系数
    YA = dot(c,meanXA);
    YB = dot(c,meanXB);
    Yc =(M*YA + N*YB)/(M+N);
    Y0 = dot(c,SampX);    %判别样本的类别
    if YA > YB
        if Y0 > Yc
            p = 1;
            disp('样本属于第一类');
        else
            if Y0 == Yc
                p = 0 ;
                disp('没法判断');
            else
                p = 2;
                disp('样本属于第二类');
            end
        end
    else
        if YA < YB
            if Y0 > Yc
                p = 2;
                disp('样本属于第二类');
            else
                if Y0 == Yc
                    p = 0 ;
                    disp('没法判断');
                else
                    p = 1;
                    disp('样本属于第一类');
                end
            end
```

```
    else
        disp('没法判断');
    end
end
```

例 17-5 Fisher 两类判别法应用实例。已知样本如下表所示：

类 别	样 本	成 分			
		x_1	x_2	x_3	x_4
A 类	1	13.58	2.79	7.8	49.6
	2	22.31	4.67	12.31	47.8
	3	28.82	4.63	16.18	62.15
	4	15.29	3.54	7.58	43.2
	5	28.29	4.90	16.12	58.7
B 类	1	2.18	1.06	1.22	20.6
	2	3.85	0.80	4.06	47.1
	3	11.4	0	3.50	0
	4	3.66	2.42	2.14	15.1
	5	12.10	0	5.68	0

用 Fisher 判别法判别样本 $X^0=[7.90 \quad 2.40 \quad 4.30 \quad 33.2]$ 和 $X^1=[12.40 \quad 5.10 \quad 4.48 \quad 24.6]$ 分别属于哪一类。

解： 在 MATLAB 窗口输入下列命令：

```
>> XA = [13.58  2.79      7.8      49.6;
22.31   4.67      12.31  47.8;
28.82   4.63      16.18  62.15;
15.29   3.54      7.58   43.2;
28.29   4.90      16.12  58.7];
>> XB = [2.18   1.06        1.22  20.6
3.85    0.80     4.06    47.1
11.4    0     3.50    0
3.66    2.42      2.14   15.1
12.10   0     5.68   0];
>> X0 = [7.90  2.40   4.30   33.2];
>> X1 = [12.40  5.10   4.48   24.6];
>> k = DistgshAnalysis(XA,XB,X0)
%样本属于第二类
k =  2
>> k = DistgshAnalysis(XA,XB,X1)
%样本属于第一类
k =  1
```

所以 X^0 属于第二类，而 X^1 属于第一类。

17.4 主成分分析

主成分分析是用尽可能少的指标来反映主题的特征，但是又不会损失原来变量太多的信息。主成分分析本质上是一种对变量的降维处理，即用较少的变量代替原来的变量，而新变量是原来变量的某种组合。其算法过程介绍如下。

❶ 得到 p 个指标的 N 组样本值。

$$X=\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{Np} \end{bmatrix}$$

❷ 计算各指标的均值和标准差。

均值：$\overline{x_i}=\sum_{j=1}^{N} x_{ji}$

标准差：$s_i=\sqrt{\frac{1}{N-1}\sum_{j=1}^{N}(x_{ji}-\overline{x_i})^2}$

❸ 计算协方差矩阵 $R=[r_{ij}]$，其中

$$r_{ij}=\frac{1}{N-1}\sum_{k=1}^{N} Y_{ki}Y_{kj}$$

$$Y_{ij}=\frac{x_{ij}-\overline{x_j}}{s_j}$$

❹ 计算 R 的特征值 $\lambda_1,\lambda_2,\cdots,\lambda_p$ 和特征向量 $\mathbf{l}_i=\begin{bmatrix} l_{1i} & l_{2i} & \cdots & l_{pi}\end{bmatrix}^T$。

❺ 将特征值从大到小排列，设排列顺序为 $\lambda_1,\lambda_2,\cdots,\lambda_p$，找出 m，使得：

$$\frac{\sum_{j=1}^{m}\lambda_j}{\sum_{j=1}^{p} r_{jj}}>0.85$$

从而确定 m 个主成分，令

$$Z_i=\sum_{j=1}^{p} l_{ji}Y_j,(i=1,2,\cdots,m)$$

❻ 计算前 m 个主成分的样本值：

$$Z_{ij}=\sum_{k=1}^{p} Y_{ik}Y_{kj},(i=1,2,\cdots,N;\ j=1,2,\cdots,m)$$

在 MATLAB 中编程实现的主成分分析法的函数为：MainAnalysis

功能：对样本进行主成分分析

调用格式：[Z,m,lamda,U] = MainAnalysis(X)

其中，X：样本矩阵；

Z：主成分样本矩阵；

m：主成分个数；

lamda：主成分对应的特征值；

U：主成分的组成系数。

主成分分析法的 MATLAB 程序代码如下所示：

```
function [Z,m,lamda,U] = MainAnalysis(X)
%样本矩阵：X
%主成分样本矩阵：Z
%主成分个数：m
%主成分对应的特征值：lamda
%主成分的组成系数：U
format long;
sz = size(X);
N = sz(1);          %样本个数
n = sz(2);
meanX = mean(X);
sx = zeros(n,1);
Y = zeros(N,n);
for i=1:n       %计算标准差矩阵
    sx(i) = sqrt(norm(X(1:N,i))^2-2*meanX(i)*sum(X(1:N,i))+N*meanX(i)^2)
/sqrt(N-1);
end
for i=1:N
    for j=1:n
        Y(i,j) = (X(i,j) - meanX(j))/sx(j);
    end
end
for i=1:n       %计算协方差矩阵
    for j=1:n
        r(i,j) = dot(Y(:,i),Y(:,j))/(N-1);
    end
end
[v,e] = eig(r);
[sortE,turnV] = sort(diag(e),'descend');  %将特征值从大到小排序
Esum = sortE(1);
Tr = sum(diag(r));
m = 1;
while 1      %确定 m 个主成分
    if Esum/Tr > 0.85
        break;
    else
        m = m+1;
```

```
            Esum = Esum + sortE(m);
        end
    end
    lamda = sortE(1:m);
    for i=1:m   %计算主成分的样本值
        U(:,i) = v(:,turnV(i));
        Z(:,i) = Y*v(:,turnV(i));
    End
    format short;
```

例 17-6 主成分分析法应用实例。对下列数据作主成分分析。

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
16.68	26.75	31.84	18.4	53.25	55	28.83	1.75
19.7	27.56	32.94	19.2	59.82	55	32.92	2.87
15.2	23.4	32.98	16.24	46.78	65	41.69	1.53
7.29	8.97	21.3	4.76	34.39	62	39.28	1.63
29.45	56.49	40.74	43.68	75.32	69	26.68	2.14
32.93	42.7	47.98	33.87	66.46	50	32.87	2.6
25.39	37.82	36.76	27.56	68.18	63	35.79	2.43
15.05	19.49	27.21	14.21	6.13	76	35.76	1.75
19.82	28.78	33.41	20.17	59.25	71	39.13	1.83
21.13	35.2	39.16	26.52	52.47	62	35.08	1.73
16.75	28.72	29.62	19.23	55.76	58	30.08	1.52
15.83	28.03	26.4	17.43	61.19	61	32.75	1.6
16.53	29.73	32.49	20.63	50.41	69	37.57	1.31
22.24	54.59	31.05	37	67.95	63	32.33	1.57
12.92	20.82	25.12	12.54	51.07	66	39.18	1.83

解：首先建立一个文本文件 Xdata3.txt，输入上面表格中的数据，然后在 MATLAB 窗口输入下列命令：

```
>> X=load('XData3.txt')
>> [Z,m,lamda,U] = MainAnalysis(X)
Z =   0.0236    1.0216    1.4549
      0.8634    2.1991   -0.4251
     -1.6305   -0.2490   -0.8189
     -3.7465    0.9605    0.2302
      3.8857   -1.6351    0.3087
      3.8484    1.4854   -1.0192
      1.6563    0.2723   -0.9813
     -2.8244   -1.1260   -0.9378
     -0.5006   -0.7942   -1.0229
      0.7116   -0.3156   -0.2899
     -0.2498    0.3585    1.5486
     -0.6971    0.2061    1.2084
     -1.0568   -1.1980   -0.0715
```

```
    1.7813   -1.3984    1.0638
   -2.0647    0.2129   -0.2481
m =  3
lamda =  4.8520
         1.2445
         0.8703
U =  0.4337   -0.0617   -0.2582
     0.4087   -0.3419    0.1177
     0.3895    0.0121   -0.3799
     0.4211   -0.3174    0.0187
     0.3586    0.0656    0.2220
    -0.1835   -0.7247   -0.3192
    -0.2944   -0.0352   -0.6353
     0.2586    0.4976   -0.4691
```

从上面的结果可以看出，主成分分析的结果得到了三个主成分（m=3），并可以得到三个主成分的表达式如下所示：

$$Z_1 = 0.4337Y_1 + 0.4087Y_2 + 0.3895Y_3 + 0.4211Y_4 + 0.3586Y_5 - 0.1835Y_6 - 0.2944Y_7 + 0.2586Y_8$$

$$Z_2 = -0.0617Y_1 - 0.3419Y_2 + 0.0121Y_3 - 0.3174Y_4 + 0.0656Y_5 - 0.7247Y_6 - 0.0352Y_7 + 0.4976Y_8$$

$$Z_3 = -0.2582Y_1 + 0.1177Y_2 - 0.3799Y_3 + 0.0187Y_4 + 0.2220Y_5 - 0.3192Y_6 - 0.6353Y_7 - 0.4691Y_8$$

主成分的样本值即为输出变量中的 Z。

17.5 小结

本章简要介绍了数据统计分析中的几大分析方法，每种方法都有不少的分析算法，例如聚类分析除了系统聚类法外还有快速聚类法、模糊聚类法等。由于篇幅等原因，没有把每种算法一一用程序重新写过，因为现在有 SAS 和 SPSS 等现成的统计分析软件，功能非常丰富。本章的目的是让读者熟悉统计分析基本算法的程序实现方法，加深对理论的了解。

附录 A MATLAB 计算常用工具箱函数注释

本书讲述的是采用 MATLAB 编程实现科学和工程中常用的算法，对于科学计算部分涉及的主要函数以及工具箱，下面以附录的形式给出。

本附录给出了 MATLAB 提供的曲线拟合工具箱、样条工具箱、偏微分方程工具箱和最优化工具箱中函数的名称及功能，以及线性代数部分常用函数的功能，便于读者进行查询和参考。

A.1 线性代数

A.1.1 矩阵分析

函数名	功能	函数名	功能
norm	矩阵或向量的范数	null	零空间
normest	估计矩阵的 2 范数	orth	正交化
rank	矩阵的秩	rref	简化矩阵为梯形形式
det	矩阵行列式的值	subspace	两个子空间的夹角

A.1.2 线性方程

函数名	功能	函数名	功能
\ 和 /	线性方程求解	lu	LU 分解
inv	矩阵的逆	ilu	不完全的 LU 分解
cond	矩阵条件数	luinc	不完全的 LU 分解
condest	1 范条件数估计	qr	QR 分解
chol	Cholesky 分解	lsqnonneg	非负线性最小二乘
cholinc	不完全的 Cholesky 分解	pinv	伪逆
linsolve	带特殊控制的线性方程求解	lscov	已知协方差的最小二乘

A.1.3 特征值和奇异值

函数名	功能	函数名	功能
eig	特征值和特征向量	polyeig	多项式特征值问题
svd	奇异值分解	condeig	已知特征值求条件数
eigs	稀疏矩阵的特征值	hess	Hessenberg 型

续表

函数名	功能	函数名	功能
svds	稀疏矩阵的奇异值和向量	qz	广义特征值的 QZ 分解
poly	特征多项式	schur	Schur 分解

A.1.4 矩阵函数

函数名	功能	函数名	功能
expm	矩阵指数	sqrtm	矩阵平方根
logm	矩阵对数	funm	计算一般矩阵函数

A.2 曲线拟合工具箱函数

A.2.1 拟合数据预处理

函数名	功能	函数名	功能
cftool	打开 GUI 形式的曲线拟合工具箱	smooth	对数据点作平滑处理
excludedata	去除异常数据点		

A.2.2 数据拟合

函数名	功能	函数名	功能
cftool	打开 GUI 形式的曲线拟合工具箱	fittype	构造一个曲线拟合对象
fit	用指定的拟合模型对数据进行拟合	get	获取拟合选项结构体的某个字段名及其值
fitoptions	创建或修改拟合选项结构体	set	设置拟合选项结构体的某个字段的值

A.2.3 拟合类型和方法

函数名	功能	函数名	功能
argnames	曲线拟合类型（或函数）对象的输入参数名	indepnames	曲线拟合类型（或函数）对象的自变量
category	曲线拟合类型（或函数）对象的拟合类型	islinear	判断曲线拟合类型（或函数）对象是否为线性
coeffnames	曲线拟合类型（或函数）对象的系数名称	numargs	曲线拟合类型（或函数）对象的输入参数个数
dependnames	曲线拟合类型（或函数）对象的因变量	numcoeffs	曲线拟合类型（或函数）对象的拟合系数个数
feval	计算曲线拟合类型（或函数）对象	probnames	曲线拟合类型（或函数）对象的问题相关参数名称
fittype	创建一个曲线拟合类型（或函数）对象	type	曲线拟合类型（或函数）对象的名称
formula	曲线拟合类型（或函数）对象的公式		

A.2.4 曲线拟合的方法

函数名	功能	函数名	功能
argnames	曲线拟合类型（或函数）对象的输入参数名	indepnames	曲线拟合类型（或函数）对象的自变量
category	曲线拟合类型（或函数）对象的拟合类型	integrate	拟合函数的积分
cfit	创建一个曲线拟合函数对象	islinear	判断曲线拟合类型（或函数）对象是否为线性
coeffnames	曲线拟合类型（或函数）对象的系数名称	numargs	曲线拟合类型（或函数）对象的输入参数个数
coeffvalues	通过拟合得到的拟合函数的系数值	numcoeffs	曲线拟合类型（或函数）对象的拟合系数个数
confint	拟合系数的值的置信区间	plot	绘制拟合曲线图
dependnames	曲线拟合类型（或函数）对象的因变量	predint	在任意点处用拟合函数计算得到的函数值的 95%置信区间
differentiate	求取拟合函数的导数	probnames	曲线拟合类型（或函数）对象的问题相关参数名称
feval	计算曲线拟合类型（或函数）对象	probvalues	拟合函数中的与问题相关的参数值
formula	曲线拟合类型（或函数）对象的公式	type	曲线拟合类型（或函数）对象的名称

A.2.5 拟合数据后处理

函数名	功能	函数名	功能
cftool	打开 GUI 形式的曲线拟合工具箱	integrate	拟合函数的积分
coeffvalues	通过拟合得到的拟合函数的系数值	plot	绘制拟合曲线图
confint	拟合系数的值的置信区间	predint	在任意点处用拟合函数计算得到的函数值的 95%置信区间
differentiate	求取拟合函数的导数	probvalues	拟合函数中的与问题相关的参数值
feval	计算曲线拟合类型（或函数）对象		

A.2.6 信息显示与帮助

函数名	功能	函数名	功能
cflibhelp	显示库中已有的曲线拟合模型，即光滑样条模型或内插模型的相关帮助信息	datastats	显示输入数据的统计信息

A.3 样条工具箱函数

A.3.1 样条 GUI 函数

函数名	功能	函数名	功能
bspligui	在节点处生成 B 样条曲线	splinetool	用一系列方法生成各种样条曲线

A.3.2 样条构建函数

函 数 名	功 能	函 数 名	功 能
csape	生成给定约束条件下的三次样条函数	rsmak	生成有理样条函数
csapi	插值生成三次样条函数	spapi	插值生成 B 样条函数
csaps	平滑生成三次样条函数	spaps	对生成的 B 样条曲线进行光滑处理
cscvn	生成一条内插参数的三次样条曲线	spap2	用最小二乘法拟合生成 B 样条函数
getcurve	动态生成三次样条曲线	spcrv	生成均匀划分的 B 样条函数
ppmak	生成分段多项式样条函数	spmak	生成 B 样条函数
rpmak	生成有理样条函数	stmak	将函数整理成 st 形式
rscvn	分段双圆弧 Hermite 插值	tpaps	薄板平滑样条函数

A.3.3 操作样条函数

函 数 名	功 能	函 数 名	功 能
fmbrk	返回样条函数的某一部分（如断点或系数等）	fnplt	画样条曲线图
fnchg	一种形式的样条函数的改变部分	fnrfn	在样条曲线中插入断点
fncmb	对样条函数进行算术运算	fntlr	生成 tarylor 系数或 taylor 多项式
fnder	求样条函数的微分（即求导数）	fnval	计算在给定点处的样条函数值
fndir	求样条函数的方向导数	fnxtr	插值函数
fnint	求样条函数的积分	fnzeros	计算函数的零点
fnjmp	在间断点处求函数值	fn2fm	把一种形式的样条函数转化成另一种形式的样条函数
fnmin	求取函数的最小值		

A.3.4 样条曲线端点和节点处理函数

函 数 名	功 能	函 数 名	功 能
aptknt	求出用于生成样条曲线的节点数组	knt2brk	从节点数组中求得节点及其重次
augknt	在已知节点数组中添加一个或多个节点	knt2mlt	从节点数组中求得节点及其重次
aveknt	求出节点数组元素的平均值	newknt	对分段多项式样条函数进行重分布
brk2knt	增加断点数组中元素的重次	optknt	求出用于内插的最优节点数组
chbpnt	求出用于生成样条曲线的合适节点数组	sorted	求出节点数组 points 的元素在节点数组 meshpoints 中属于第几个分量

A.3.5 解线性方程组的函数

函 数 名	功 能	函 数 名	功 能
bkbrk	描述分块对角矩阵的详细情况	slvblk	解对角占优的线性方程组

A.3.6 样条及工具箱的信息显示

函数名	功能	函数名	功能
bspline	显示一条 B 样条曲线和它的分段形式	spterms	显示样条工具箱术语的解释

A.3.7 实用函数

函数名	功能	函数名	功能
franke	Franke 双变量测试函数	stcol	配置矩阵的离散变换
spcol	生成 B 样条函数的配置矩阵	subplus	返回参数的正数部分
splpp	0 左面的节点序列从 B 形式转换为 pp 形式	titanium	返回 titanium 热数据
sprpp	0 右面的节点序列从 B 形式转换为 pp 形式		

A.4 偏微分方程工具箱函数

A.4.1 偏微分方程求解算法函数

函数名	功能	函数名	功能
adaptmesh	生成自适应网格并求解 PDE 问题	parabolic	求解抛物线型 PDE 问题
assema	组合面积的整体贡献	pdeeig	求解特征值 PDE 问题
assemb	组合边界条件的贡献	pdenonlin	求解非线性 PDE 问题
assempde	组合刚度矩阵和 PDE 问题的右端项	poisolv	在矩形网格上对泊松方程进行快速求解
hyperbolic	求解双曲线 PDE 问题		

A.4.2 用户界面算法函数

函数名	功能	函数名	功能
pdecirc	绘圆	pdepoly	绘多边形
pdeellip	绘椭圆	pderect	绘矩形
pdemdlcv	将 PDE 工具箱 1.0 模型的 M 文件转换为 PDE 工具箱 1.0.2 版本的格式	pdetool	打开 PDE 工具箱图形用户集成界面（GUI）

A.4.3 几何算法函数

函数名	功能	函数名	功能
csgchk	核对几何描述矩阵的有效性	pdearcl	在参数表示和圆弧长度之间进行内插
csgdel	删除最小子域之间的界线	poimesh	在矩形几何图形上生成规则网格
decsg	将建设性实体几何模型分解为最小子域	refinemesh	加密一个三角形网格
initmesh	创建初始三角形网格	wbound	写边界条件指定文件
jigglemesh	微调三角形网格的内部点	wgeom	写几何指定函数

A.4.4 绘图函数

函数名	功能	函数名	功能
pdecont	绘等值线图	pdeplot	一般 PDE 工具箱绘图函数
pdegplot	绘制 PDE 几何图	pdesurf	绘三维表面图
pdemesh	绘 PDE 三角形网格		

A.4.5 实用算法函数

函数名	功能	函数名	功能
dst, idst	离散化 sin 转换	pdesde, pdesdp, pdesdt	子域集合中点/边缘/三角形的指数
pdeadgsc	使用相对容限临界值选择三角形	pdesmech	计算结构力学张量函数
pdeadworst	选择相对于最坏值的三角形	pdetrg	三角形几何数据
pdecgrad	PDE 解的变动	pdetriq	三角形质量度量
pdeent	与给定三角形集合相邻的三角形的指数	poiasma	用于泊松方程快速求解器的边界点矩阵
pdegrad	PDE 解的梯度	poicalc	矩形网格上泊松方程的快速求解器
pdeintrp	从节点数据至三角形中点数据进行内插	poiindex	经过规范排序的矩形网格的点的指数
pdejmps	对于自适应网格进行误差估计	sptarn	求解广义稀疏特征值问题
pdeprtni	从三角形中点数据向节点数据进行内插	tri2grid	从 PDE 三角形网格到矩形网格进行内插

A.4.6 自定义算法函数

函数名	功能	函数名	功能
pdebound	边界条件 M 文件	pdegeom	几何模型 M 文件

A.5 最优化工具箱函数

A.5.1 最小化函数

函数名	功能	函数名	功能
bintprog	求解 0-1 整数规划问题	fminunc	无约束多变量问题最小化
fgoalattain	多目标达到问题	fseminf	半无限约束多变量非线性问题最小化
fminbnd	有边界的标量非线性最小化	ktrlink	使用第三方库 KNITRO®求解多变量非线性问题最小化
fmincon	有约束的非线性最小化	linprog	求解线性规划问题
fminimax	求解最大最小约束问题	quadprog	求解二次规划问题
fminsearch	使用无导数方法求解无约束多变量问题最小化		

A.5.2 方程求解函数

函数名	功　能	函数名	功　能
fsolve	非线性方程求解	fzero	单变量连续函数求根

A.5.3 最小二乘函数

函数名	功能	函数名	功能
lsqcurvefit	非线性曲线拟合	lsqnonlin	非线性最小二乘
lsqlin	有约束线性最小二乘	lsqnonneg	非负线性最小二乘

A.5.4 图形用户界面（GUI）

函数名	功能
optimtool	选择求解器、最优化选项和求解最优化问题的工具

A.5.5 实用函数

函数名	功　能	函数名	功　能
color	稀疏有限差分的列分区	optimget	获取优化选项结构的参数值
fzmult	零空间基的乘法	optimset	创建或编辑优化选项结构

附录B 本书所编写的算法程序索引

本书精选了科学和工程中常用的 200 余个算法，并编写了这些算法的 MATLAB 程序。这些程序编写思路清晰，注释丰富，而且经过了验证，可以直接应用于实际。建议读者从一开始就耐心地分析、练习和调试这些程序，在实践中掌握 MATLAB 编程。

这些程序代码在随书光盘中有序地列出，读者可方便地进行应用。同时，为了便于读者阅读、分析、学习和应用这些算法程序，本附录给出了这些算法程序在书中的索引。

B.1 第 4 章“插值”的算法程序索引

函数名	功能	位于的章节	位于的页码
Language	求已知数据点的拉格朗日插值多项式	4.1	68
Atken	求已知数据点的艾特肯插值多项式	4.2	70
Newton	求已知数据点的均差形式的牛顿插值多项式	4.3	72
Newtonforward	求已知数据点的前向牛顿差分插值多项式	4.4.1	75
Newtonback	求已知数据点的后向牛顿差分插值多项式	4.4.1	75
Gauss	求已知数据点的高斯插值多项式	4.4.2	79
Hermite	求已知数据点的埃尔米特插值多项式	4.5	84
SubHermite	求已知数据点的分段三次埃尔米特插值多项式及其插值点处的值	4.6	85
SecSample	求已知数据点的二次样条插值多项式及其插值点处的值	4.7.1	87
ThrSample1	求已知数据点的第一类三次样条插值多项式及其插值点处的值	4.7.2	89
ThrSample2	求已知数据点的第二类三次样条插值多项式及其插值点处的值	4.7.2	89
ThrSample3	求已知数据点的第三类三次样条插值多项式及其插值点处的值	4.7.2	89
BSample	求已知数据点的第一类 B 样条的插值	4.7.3	97
DCS	用倒差商算法求已知数据点的有理分式形式的插值分式	4.8	100
Neville	用 Neville 算法求已知数据点的有理分式形式的插值分式	4.8	100
FCZ	用倒差商算法求已知数据点的有理分式形式的插值分式	4.9	104
DL	用双线性插值求已知点的插值	4.10.1	108
DTL	用二元三点拉格朗日插值求已知点的插值	4.10.2	110
DH	用分片双三次埃尔米特插值求插值点的 z 坐标	4.10.3	112

B.2 第 5 章“函数逼近”的算法程序索引

B.3 第 6 章“矩阵特征值计算”的算法程序索引

B.4 第 7 章“数值微分”的算法程序索引

B.5 第 8 章“数值积分”的算法程序索引

B.6 第 9 章“方程求根”的算法程序索引

续表

函 数 名	功　　能	位于的章节	位于的页码
NewtonRoot	用牛顿法求方程的一个根	9.2.10	252
SimpleNewton	用简化牛顿法求方程的一个根	9.2.10	254
NewtonDown	用牛顿下山法求方程的一个根	9.2.10	255
YSNewton	逐次压缩牛顿法求多项式的全部实根	9.2.11	257
Union1	用联合法 1 求方程的一个根	9.2.12	258
TwoStep	用两步迭代法求方程的一个根	9.2.13	262
Montecarlo	用蒙特卡洛法求方程的一个根	9.2.14	264
MultiRoot	求存在重根的方程的一个重根	9.2.15	265

B.7 第 10 章“非线性方程组求解”的算法程序索引

函 数 名	功　　能	位于的章节	位于的页码
mulStablePoint	用不动点迭代法求非线性方程组的一个根	10.1	267
mulNewton	用牛顿法法求非线性方程组的一个根	10.2	268
mulDiscNewton	用离散牛顿法法求非线性方程组的一个根	10.3	271
mulMix	用牛顿-雅可比迭代法求非线性方程组的一个根	10.4.1	274
mulNewtonSOR	用牛顿-SOR 迭代法求非线性方程组的一个根	10.4.2	276
mulDNewton	用牛顿下山法求非线性方程组的一个根	10.5	279
mulGXF1	用两点割线法的第一种形式求非线性方程组的一个根	10.6	280
mulGXF2	用两点割线法的第二种形式求非线性方程组的一个根	10.6	282
mulVNewton	用拟牛顿法求非线性方程组的一组解	10.7	284
mulRank1	用对称秩 1 算法求非线性方程组的一个根	10.8	286
mulDFP	用 D-F-P 算法求非线性方程组的一组解	10.9	287
mulBFS	用 B-F-S 算法求非线性方程组的一个根	10.10	289
mulNumYT	用数值延拓法求非线性方程组的一组解	10.11	291
DiffParam1	用参数微分法中的欧拉法求非线性方程组的一组解	10.12	293
DiffParam2	用参数微分法中的中点积分法求非线性方程组的一组解	10.12	294
mulFastDown	用最速下降法求非线性方程组的一组解	10.13	296
mulGSND	用高斯牛顿法求非线性方程组的一组解	10.14	298
mulConj	用共轭梯度法求非线性方程组的一组解	10.15	299
mulDamp	用阻尼最小二乘法求非线性方程组的一组解	10.16	301

B.8 第 11 章“解线性方程组的直接法”的算法程序索引

函 数 名	函数功能	位于的章节	位于的页码
SolveUpTriangle	求上三角系数矩阵的线性方程组的解	11.2	305
GaussXQByOrder	高斯顺序消去法求线性方程组的解	11.2.1	306
GaussXQLineMain	高斯按列主元消去法求线性方程组的解	11.2.2	308
GaussXQAllMain	高斯全主元消去法求线性方程组的解	11.2.2	311
GaussJordanXQ	高斯-若当消去法求线性方程组的解	11.2.3	313
Crout	克劳特分解法求线性方程组的解	11.3.1	316
Doolittle	多利特勒分解法求线性方程组的解	11.3.2	318

续表

函 数 名	函数功能	位于的章节	位于的页码
SymPos1	LL^T 分解法求线性方程组的解	11.4.1	320
SymPos2	LDL^T 分解法求线性方程组的解	11.4.2	322
SymPos3	改进的 LDL^T 分解法求线性方程组的解	11.4.3	323
followup	追赶法求线性方程组的解	11.5	325
InvAddSide	加边求逆法求线性方程组的解	11.6.1	327
Yesf	叶尔索夫求逆法求线性方程组的解	11.6.2	329
qrxq	QR 分解法求线性方程组的解	11.7	331

B.9 第 12 章“解线性方程组的迭代法”的算法程序索引

函 数 名	功　　能	位于的章节	位于的页码
rs	理查森迭代法求线性方程组的解	12.1.1	334
crs	理查森参数迭代法求线性方程组的解	12.1.1	336
grs	理查森迭代法求线性方程组的解	12.1.2	338
jacobi	雅可比迭代法求线性方程组的解	12.1.3	339
gauseidel	高斯-赛德尔迭代法求线性方程组的解	12.1.4	341
SOR	超松弛迭代法求线性方程组的解	12.1.5	343
SSOR	对称逐次超松弛迭代法求线性方程组的解	12.1.5	344
JOR	雅可比超松弛迭代法求线性方程组的解	12.1.6	346
twostep	两步迭代法求线性方程组的解	12.1.7	348
fastdown	最速下降法求线性方程组的解	12.1.8	350
conjgrad	共轭梯度法求线性方程组的解	12.1.8	352
preconjgrad	预处理共轭梯度法求线性方程组的解	12.1.8	353
BJ	块雅克比迭代法求线性方程组的解	12.1.9	357
BGS	块高斯-赛德尔迭代法求线性方程组的解	12.1.9	359
BSOR	块逐次超松弛迭代法求线性方程组的解	12.1.9	362

B.10 第 13 章“随机数生成”的算法程序索引

函 数 名	功　　能	位于的章节	位于的页码
PFQZ	用平方取中法产生随机数列	13.1	365
MixMOD	用混合同余法产生随机数列	13.2.1	367
MulMOD1	用乘同余法 1 产生随机数列	13.2.2	370
MulMOD2	用乘同余法 2 产生随机数列	13.2.2	371
PrimeMOD	用素数模同余法产生随机数列	13.2.3	372
PowerDist	产生指数分布的随机数列	13.3	374
LaplaceDist	产生拉普拉斯分布的随机数列	13.4	376
RelayDist	产生瑞利分布的随机数列	13.5	377
CauthyDist	产生柯西分布的随机数列	13.6	379
AELDist	产生爱尔朗分布的随机数列	13.7	380
GaussDist	产生正态分布的随机数列	13.8	381
WBDist	产生韦伯分布的随机数列	13.9	384

续表

函 数 名	功　　能	位于的章节	位于的页码
PoisonDist	产生泊松分布的随机数列	13.10	385
BenuliDist	产生贝努里分布的随机数列	13.11	387
BGDist	产生贝努里-高斯分布的随机数列	13.12	388
TwoDist	产生二项式分布的随机数列	13.13	389

B.11 第 14 章“特殊函数计算”的算法程序索引

函 数 名	功　　能	位于的章节	位于的页码
gamafun	用逼近法计算伽玛函数的值	14.1	391
lngama	用 Lanczos 算法计算伽玛函数的自然对数值	14.1	393
Beta	用伽玛函数计算贝塔函数的值	14.1	394
gamap	用逼近法计算不完全伽玛函数的值	14.2	396
betap	用逼近法计算不完全贝塔函数的值	14.3	398
bessel	用逼近法计算伽玛函数的值	14.4	402
bessel2	用逼近法计算第二类整数阶贝塞尔函数值	14.5	407
besselm	用逼近法计算变型的第一类整数阶贝塞尔函数值	14.6	412
besselm2	用逼近法计算变型的第二类整数阶贝塞尔函数值	14.7	416
ErrFunc	用高斯积分计算误差函数值	14.8	420
SIx	用高斯积分计算正弦积分值	14.9	422
CIx	用高斯积分计算余弦积分值	14.9	423
EIx	用高斯积分计算指数积分值	14.9	424
EIx2	用逼近法计算指数积分值	14.9	425
Ellipint1	用高斯积分计算第一类椭圆积分值	14.10	426
Ellipint2	用高斯积分计算第二类椭圆积分值	14.11	427

B.12 第 15 章“常微分方程的初值问题”的算法程序索引

函 数 名	功　　能	位于的章节	位于的页码
DEEuler	用欧拉法求一阶常微分方程的数值解	15.1.1	429
DEimpEuler	用隐式欧拉法求一阶常微分方程的数值解	15.1.2	431
DEModifEuler	用改进欧拉法求一阶常微分方程的数值解	15.1.3	433
DELGKT2_mid	用中点法求一阶常微分方程的数值解	15.2.1	435
DELGKT2_suen	用休恩法求一阶常微分方程的数值解	15.2.1	436
DELGKT3_suen	用休恩三阶法求一阶常微分方程的数值解	15.2.2	438
DELGKT3_kuta	用库塔三阶法求一阶常微分方程的数值解	15.2.2	439
DELGKT4_lungkuta	用经典龙格-库塔法求一阶常微分方程的数值解	15.2.3	440
DELGKT4_jer	用基尔法求一阶常微分方程的数值解	15.2.3	441
DELGKT4_qt	用变形龙格-库塔法求一阶常微分方程的数值解	15.2.3	443
DELSBRK	用罗赛布诺克半隐式法求一阶常微分方程的数值解	15.2.4	445
DEMS	用默森单步法求一阶常微分方程的数值解	15.3	447
DEMiren	用米尔恩法求一阶常微分方程的数值解	15.4	449
DEYDS	用亚当斯法求一阶常微分方程的数值解	15.4	450

续表

B.13 第 16 章“偏微分方程的数值解法”的算法程序索引

B.14 第 17 章“数据统计和分析”的算法程序索引

参考文献

[1] 《现代应用数学手册》编委会. 现代应用数学手册——计算与数值方法卷. 北京: 清华大学出版社，2005
[2] 余锦华，杨维权. 多元统计分析与应用. 广州: 中山大学出版社，2005
[3] 王正林，刘明. 精通 MATLAB 7. 北京: 电子工业出版社，2006
[4] 何晓群. 现代统计分析方法与应用（第二版）. 北京: 中国人民大学出版社，2007
[5] John H.Mathews, Kurtis D.Fink 著，周璐，陈渝，钱方等译. 数值方法（MATLAB 版）第四版. 北京: 电子工业出版社，2005
[6] 郑惠娆，陈绍林，莫忠息，黄象鼎. 数值计算方法. 武汉: 武汉大学出版社，2002
[7] 李庆扬，关治，白峰杉. 数值计算原理. 北京: 清华大学出版社，2000
[8] 王正林，龚纯，何倩. 精通 MATLAB 科学计算. 北京: 电子工业出版社，2007
[9] 何光渝. Visual C++常用数值算法集. 北京: 科学出版社，2002
[10] 何渝. 计算机常用数值算法与程序（C++版）. 北京: 人民邮电出版社，2003
[11] 陆金甫，关治. 偏微分方程数值解法. 北京: 清华大学出版社，2003